#135 JANUARY 1988

2.95 (3.95 CANADA)

Dr. Dobb's Journal of Software Tools

FOR THE PROFESSIONAL PROGRAMMER

The 68030
How Hot Is It?

Focus on Macintosh Programming

New Mac Column

Languages:

0 38351 16562 8 01

ter than ever before!

4.0 uses logical units for separate compilation

Pascal 4.0 lets you break up the code gang into "units," or "chunks." These logical modules can be worked with swiftly and separately—so that an error in one module is seeable and fixable, and you're not sent through all your code to find one error. Compiling and linking these separate units happens in a flash because your compiling horsepower is better than 27,000 lines a minute.* And 4.0 also includes an automatic project Make.

4.0's cursor automatically lands on any trouble spot

4.0's interactive error detection and location means that the cursor automatically lands where the error is. While you're compiling or running a program, you get an error message at the top of your screen *and* the cursor flags the error's location for you.

4.0 gives you an integrated programming environment

4.0's integrated environment includes pull-down menus and a built-in editor. Your program output is automatically saved and shown in the output window. You can Scroll, Pan, or Page through all your output and know where everything is all the time. Given 4.0's integration, you can edit, compile, find and correct errors—all from inside the integrated development environment.

You'll never lose your mind, because 4.0 never loses your place

Whenever you re-load 4.0, it remembers what you and it were doing before you left. It puts you right back in the editor with the same file and in the same place as you were working last.

*Run on an 8 MHz IBM AT.

**If within 60 days of purchase this product does not perform in accordance with our claims, call our customer service department, and we will arrange a refund.

Please check box(es)	*Sugg. Retail*	*Upgrade Price†*	Serial No.
☐ Turbo Pascal 4.0 Compiler	*$ 99.95*	*$ 39.95*	________
☐ Turbo Pascal Tutor	*69.95*	*19.95*	________
☐ Turbo Pascal Database Toolbox	*99.95*	*29.95*	________
☐ Turbo Pascal Graphix Toolbox	*99.95*	*29.95*	________
☐ Turbo Pascal Editor Toolbox	*99.95*	*29.95*	________
☐ Turbo Pascal Numerical Methods Toolbox	*99.95*	*29.95*	________
☐ Turbo Pascal Gameworks	*99.95*	*29.95*	________

Total product amount $ ________

CA and MA residents add sales tax $ ________

In US please add $5 shipping and handling for each product ordered $ ________

Outside US please add $10 shipping and handling for each product ordered $ ________

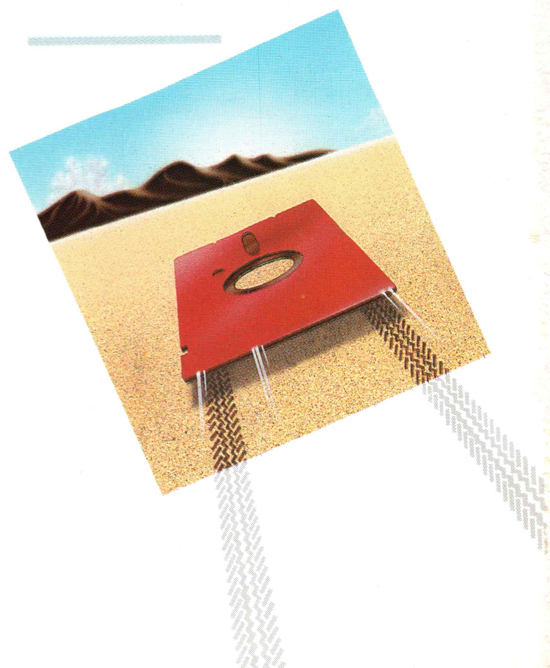
Total amount enclosed $ ________

Please specify diskette size: ☐ 5¼" ☐ 3½"

Payment: ☐ VISA ☐ MC ☐ Check ☐ Bank Draft

Credit card expiration date: ________/________

Card # |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|



†To qualify for the upgrade price you must give the serial number of the equivalent product you are upgrading.

The fast lane is *fas*

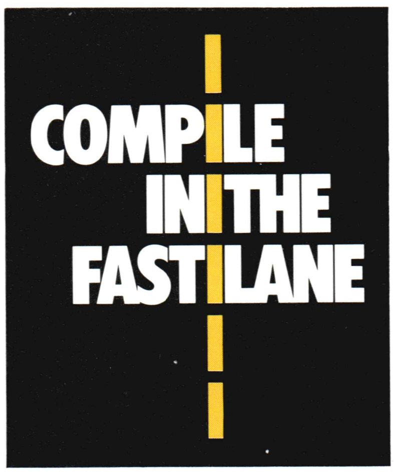
Our new Turbo Pascal® 4.0 is so fast, it's almost reckless. How fast? Better than 27,000 lines of code per minute. That's much faster than 3.0 or any other Pascal compiler and the reason why you need 4.0 today.

Pascal. The fastest and the best.

If you're just now learning a computer language, learn Pascal. If you're already programming in Pascal, you're programming with a winner because Pascal is the worldwide language of choice. Pascal is the most popular language in university computer science classes and with computer enthusiasts who appreciate Pascal's modern programming structure. It's powerful, coherent, easy to learn and use—and with Turbo Pascal 4.0—faster than ever before.

Turbo Pascal: Technical excellence

Commitment to technical excellence and



superiority also means commitment to detail, however painstaking, and that takes time. 4.0's predecessor, Turbo Pascal 3.0 is the worldwide standard, and with Turbo Pascal 4.0, we've bettered that standard. 4.0 is clearly the world's fastest development tool for the IBM® PS/2 series, PC's and compatibles—and the world's favorite Pascal compiler.

4.0 breaks the code barrier

No more swapping code in and out to beat the 64K code barrier. Designed for large programs, Turbo Pascal 4.0 lets you use all 640K memory in your computer. You paid for all that memory, now you can use it freely.

For the IBM PS/2 and the IBM and Compaq families of personal computers and all 100% compatibles.

YES! **I want to upgrade to Turbo Pascal 4.0 and the 4.0 Toolboxes**

Registered owners have been notified by mail. If you are a registered Turbo Pascal user and have not been notified of Version 4.0 by mail, please call us at *(800) 543-7543*. To upgrade if you have not registered your product, just send the original registration form from your manual and payment with this completed coupon to:

Pascal 4.0 Upgrade Dept.
Borland International
4585 Scotts Valley Drive
Scotts Valley, CA 95066

Name ______________________

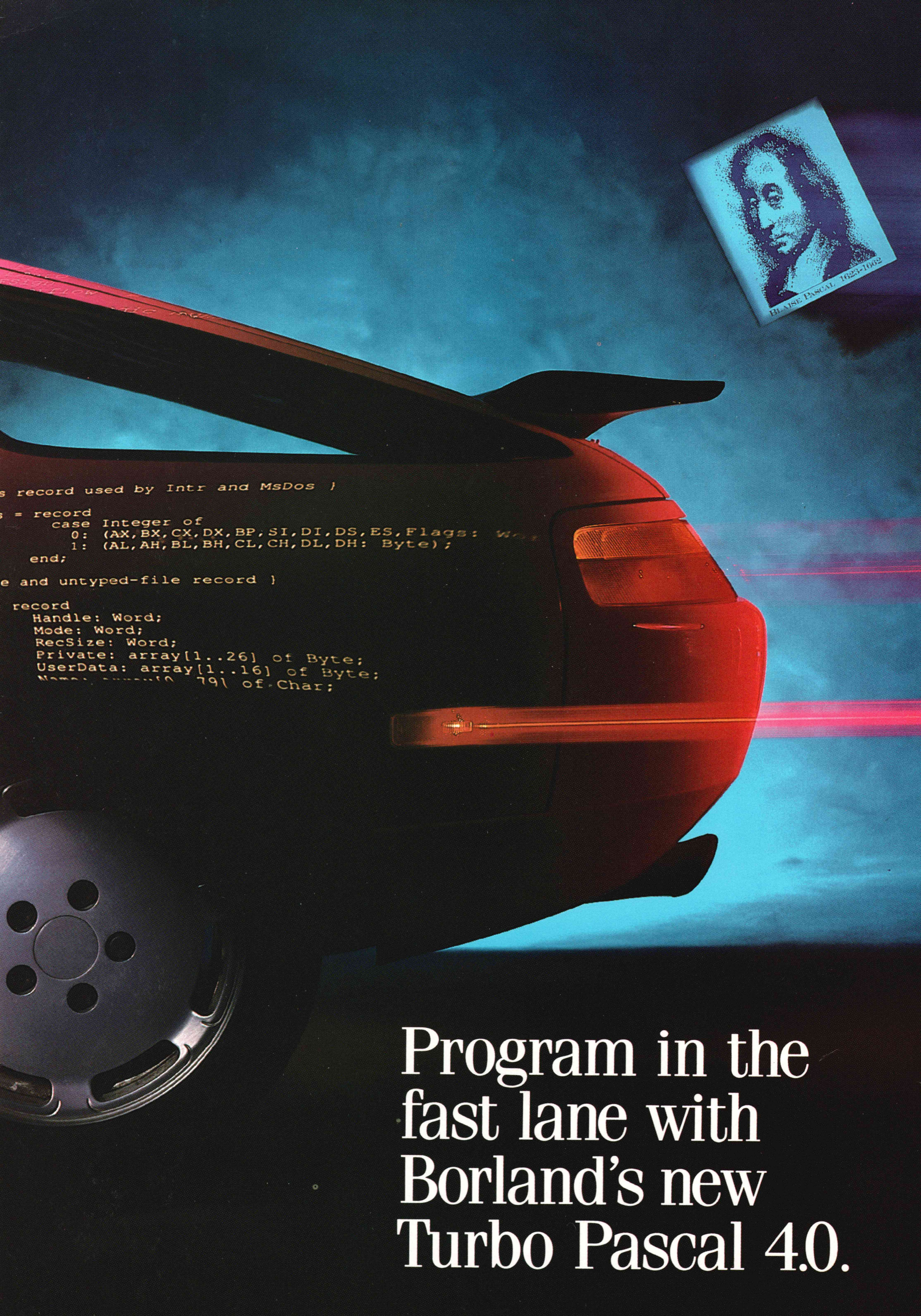
Ship Address ______________________

City ______________ State ______

Zip ____________ Telephone (   ) ____________

This offer is limited to one upgrade per valid registered product. It is good until June 30, 1988. Not good with any other offer from Borland.
Outside U.S. make payments by bank draft payable in U.S. dollars drawn on a U.S. bank. CODs and purchase orders will not be accepted by Borland.

DDJ 1 88


Blaise Pascal 1623-1662
s record used by Intr and MsDos }
s = record
case Integer of
0: (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags:
1: (AL,AH,BL,BH,CL,CH,DL,DH: Byte);
end;
e and untyped-file record }
record
Handle: Word;
Mode: Word;
RecSize: Word;
Private: array[1..26] of Byte;
UserData: array[1..16] of Byte;
Program in the fast lane with Borland's new Turbo Pascal 4.0.

Now's the time for a *fast* decision: Upgrade now to 4.0!

Compatibility with Turbo Pascal 3.0

We've created 4.0 to be highly compatible with version 3.0 and included a conversion program and compatibility units to help you convert all your 3.0 programs to 4.0.

Highlights of Borland's new Turbo Pascal 4.0

- Compiles 27,000 lines per minute
- Supports >64K programs
- Uses units for separate compilation
- Integrated development environment
- Interactive error detection/location
- Includes a command line version of the compiler

4.0 also

- Saves output screen in a window
- Supports 25, 43 and 50 lines per screen
- Generates MAP files for debugging
- Has graph units including CGA, EGA, VGA, MCGA, 3270 PC, AT & T 6300 & Hercules support
- Supports extended data types (including word, long integers)
- Does smart linking
- Comes with a free revised MicroCalc spreadsheet source code

4.0 is all yours for only $99.95

Sieve (25 iterations)

	Turbo Pascal 4.0	Turbo Pascal 3.0
Size of Executable File	2224 bytes	11682 bytes
Execution speed	9.3 seconds	9.7 seconds

Sieve of Eratosthenes, run on an 8MHz IBM AT

Since the source file above is too small to indicate a difference in compilation speed we compiled our GOMOKU program from Turbo Gameworks to give you a true sense of how much faster 4.0 really is!

Compilation of GO.PAS (1006 lines)

	Turbo Pascal 4.0	Turbo Pascal 3.0
Compilation speed	2.2 seconds	3.6 seconds
Lines per minute	27,436	16,750

GO.PAS compiled on an 8 MHz IBM AT

60-Day Money-Back Guarantee**

BORLAND INTERNATIONAL

For the dealer nearest you or to order call
(800) 543-7543.

CIRCLE NO. 101 ON READER SERVICE CARD

BORLAND

Now, just $129 and supports Microsoft C 5.0 and QuickC

Blaise puts the Accent on C with C TOOLS PLUS/5.0™

Enhance your Microsoft C programming environment with C TOOLS PLUS/5.0™—a new, quintessential library of C functions. C TOOLS PLUS/5.0 from Blaise Computing Inc. puts a prime accent on quickly building professional applications using the full power of Microsoft C Version 5.0 and QuickC. Now you can concentrate on program creativity by having full control over DOS, menus, interrupt service routines, memory resident programs, printer and keyboard control, and more!

C TOOLS PLUS/5.0 prebuilt libraries are ready to use with either QuickC or the Microsoft C Version 5.0 command line environment. Complete documented source code is included so that you can study and adapt it to your specific needs. Blaise Computing's attention to detail, like the use of full function prototyping, cleanly organized header files, and a comprehensive, fully-indexed manual, makes C TOOLS PLUS/5.0 the choice for experienced developers as well as newcomers to C.

Continuous refinement of Blaise Computing's library products has produced a collection of tools that are unsurpassed for reliability, functionality and ease of use. Built upon the widely acclaimed C TOOLS PLUS, C TOOLS PLUS/5.0 includes such highly-developed features as:

◆ **WINDOWS**
- Stackable, removable.
- Optional borders, cursor memory.
- Accept user input, formatted output.
- "printf" window-oriented output. **NEW!**

◆ **INTERRUPT SERVICE ROUTINES**
- Capture DOS critical errors and keystrokes.
- Install hardware interrupt handlers.

◆ **RESIDENT SOFTWARE SUPPORT**
- Install, detect and remove memory resident programs.

◆ **MENUS**
- Horizontal and pulldown. **NEW!**
- Lotus-style support. **NEW!**

◆ **INTERVENTION CODE**
- Schedule C functions at specified times, intervals or with a "hot key." **NEW!**
- Take full advantage of DOS, even from memory resident programs. **NEW!**

◆ **FAST DIRECT VIDEO ACCESS**
- All monitors, even EGA 43-line mode.

◆ **PRINTER CONTROL**
- Access BIOS print functions. **NEW!**
- Control the DOS PRINT utility. **NEW!**

◆ **UTILITIES AND MACROS**
- Take advantage of DOS file structure.
- Manipulate data types, far & near pointers. **NEW!**
- Access any memory areas with fast "peek" and "poke" macros. **NEW!**

C TOOLS PLUS/5.0 supports the Microsoft C Version 5.0 and QuickC compilers, requires DOS 2.00 or later and is just **$129.00.**

C ASYNCH MANAGER™ Version 2.0 IMPROVED!

C ASYNCH MANAGER is a library of functions designed to help you incorporate asynchronous communication capabilities into your application programs. Version 2.0 has been rewritten especially for Microsoft C Version 5.0 and Borland's Turbo C. Simultaneous buffered input and output to both COM ports at speeds up to 9600 baud, XON/XOFF protocol, modem control and XMODEM file transfer are among the many features supported and is priced at just **$175.00.**

Blaise computing Inc. has a full line of support products for both Pascal and C. Call today for your free information packet.

BLAISE COMPUTING INC.
2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

THE BLAISE MENU

VIEW MANAGER **$275.00**
General screen control; paint screens; block mode data entry or field-by-field control with instant screen access. For C or MS-Pascal.

Turbo C TOOLS **$129.00**
Windows; ISRs; intervention code; screen handling and EGA 43-line text mode support; direct screen access; DOS file handling and more. For Turbo C.

Turbo POWER SCREEN
COMING SOON! General screen management; paint screens; block mode data entry or field-by-field control with instant screen access. For Turbo Pascal.

Turbo POWER TOOLS PLUS **$129.00**
Screen and window management including EGA support; DOS memory control; ISRs; scheduled intervention code; and much more. Now supports Turbo Pascal 4.0!

Turbo ASYNCH PLUS **$129.00**
Interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF; up to 9600 baud; modem and XMODEM control. Now supports Turbo Pascal 4.0!

PASCAL TOOLS/TOOLS 2 **$175.00**
Expanded string and screen handling; graphics routines; memory management; general program control; DOS file support and more. For MS-Pascal.

ASYNCH MANAGER **$175.00**
Full featured interrupt driven support for the COM ports. I/O buffers up to 64K; XON/XOFF; up to 9600 baud; modem control and XMODEM. For MS-Pascal.

KeyPlayer **$49.95**
"Super-batch" program. Create batch files which can invoke programs and provide input to them; run any program unattended; create demonstration programs; analyze keyboard usage.

EXEC **$95.00**
NEW VERSION! Program chaining executive. Chain one program from another in different languages; specify common data areas; less than 2K of overhead.

RUNOFF **$49.95**
Text formatter for all programmers; flexible printer control; user-defined variables; index generation; general macro facility. Crafted in Turbo Pascal.

LIGHT TOOLS **$99.95**
Windows; ISRs; EGA 43-line text mode; direct screen access; DOS file handling and more. For the Datalight C compiler.

TO ORDER CALL TOLL FREE
800-333-8087
TELEX NUMBER-338139

Yes! Send me the prime accent!
Enclosed is $ ________ for ________ copies of ________.
☐ Please send me more information on your products.
CA residents add Sales Tax. Domestic orders add $4.00 for UPS shipping, $10.00 for Federal Express standard air.
________ Phone: (____) ____-____
Name: ________
Address: ________ State: ________ Zip: ________
City: ________ Exp. Date: ________
VISA or MC#: ________

Microsoft is a registered trademark of Microsoft Corporation. QuickC is a trademark of Microsoft Corporation. Turbo C is a registered trademark of Borland International.

JANUARY 1988 VOLUME 13, ISSUE 1

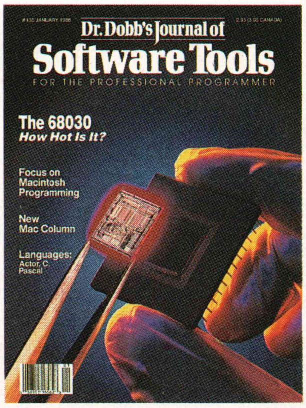
CONTENTS

ARTICLES

COLUMNS

FORUM

PROGRAMMER'S SERVICES



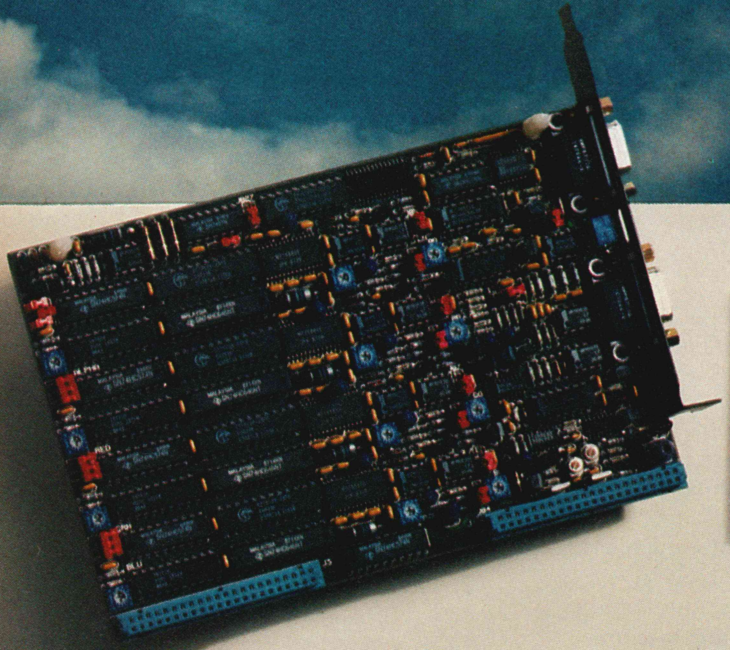
About the Cover

The Motorola 68030. Two googolplex of transistors, count them. Is it the Next Big Thing?

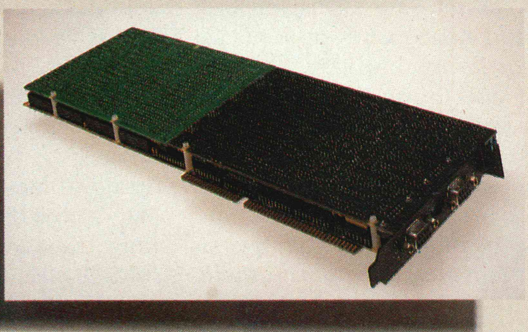
Next Issue

We don't normally comment on unreleased products, but since you are the soul of discretion, we'll tell you this much: the February release (Version 13.2) is now in alpha and will feature tools for debugging. Your SDK should arrive in 30 days and will include reviews of debugging tools and articles on debugging and a new section of short reviews called The Examining Room. The user interface will show minor bug fixes, and the delivery medium will again be high-bandwidth paper. And of course it's all upwardly compatible with the preceding 135 releases.

Introducing VISTA™ Videographics.



VISTA is a powerful single-slot videographics adapter which captures and displays video signals in real time.



Let's Get Specific.

We knew you couldn't resist seeing the facts, and frankly, our engineers wouldn't have it any other way. Here is an overview of VISTA's key features.

FEATURES:

- 4Mbytes of Video RAM on-board
- Texas Instruments' TMS 34010 GSP
- Flexible, programmable resolutions
- NTSC and PAL compatible
- Four 8-bit channels for real-time capture
- Fully integrated genlock
- Processor memory expandable in 2Mbyte increments to 12Mbytes
- Four 2K x 8-bit CMOS static RAM LUTs
- Display can be color-mapped, RGB, or a versatile combination of both
- Interlaced and non-interlaced display
- Binary and fractional programmable zoom capability, creates horizontal and vertical magnify or minify
- Smooth horizontal and vertical programmable panning, includes wrap-around and split screen
- Suggested Retail Price: $5995.

ADDRESSABLE RESOLUTIONS:

32 bits/pixel	16 bits/pixel	8 bits/pixel
1024x1024	2048x1024	4096x1024
512x2048	1024x2048	2048x2048
256x4096	512x4096	1024x4096

CAPTURE RESOLUTIONS:*

NTSC	PAL
(RS-170A)	(CCIR-624)
756x486	738x576
604x486	590x576
504x486	492x576
432x486	422x576

Resolutions are programmable; these are nominal ones for interlaced NTSC and PAL compatible.

DISPLAY RESOLUTIONS:*

NTSC	PAL	Interlaced	Non-Interlaced
(RS-170A)	(CCIR-624)		
1512x486	1476x576	1024x768	768x576
1008x486	984x576	(60 Hz)	(50 Hz)
756x486	738x576		
604x486	590x576	768x768	756x486
504x486	492x576	(80 Hz)	(60 Hz)

Resolutions are programmable; these are nominal ones.

COMPUTER REQUIREMENTS:

Host Type:	IBM PC AT and 100% Compatibles, Compaq 386, Apollo DN 3000-single-slot board
Data Bus:	16-bit or 8-bit (self-configuring)
Bus Clock:	6MHz to 12MHz
Power Consumption:	15 Watts

It's So Flexible, We've Added Support.

With its Texas Instruments TMS 34010 graphics processor, large quantity of video memory, and proprietary video cross-point, VISTA can be programmed for an array of powerful market-specific videographic applications. To help you maximize VISTA's potential, Truevision offers a range of C-language programming tools for developers. And when your system is market-ready, we'll support your marketing efforts with our *TRUEVISION SOFTWARE CATALOG, TRUEVISION NEWS,* and *THE PULSE.*

We're For Higher

Resolution…Power…Flexibility… Quality. Join the many key manufacturers and developers already working with the state of the videographics art, VISTA. Call us at **800/858-TRUE** for more information on the VISTA Developer's Program. We're ready to take your application today.

AT&T
Electronic Photography and Imaging Center
7351 Shadeland Station, Suite 100
Indianapolis, IN 46256
800/858-TRUE

AT&T | TRUEVISION®

CIRCLE NO. 103 ON READER SERVICE CARD

IBM is a registered trademark of International Business Machines Corp. **International Inquiries:** contact *Techexport* at 617/890-6507 (USA), or London at 44-1-991-0121. In Italy, contact *SIRIO Informatica* at 39-2-301-0051. Suggested retail price is US domestic price.

EDITORIAL

Back in the late 1970s my best friend Renèe gave me a pocket calculator for my birthday. Needless to say, I was impressed. I was sure this modern gadget would make life a lot easier. It was a great little machine: ran on a 9-volt battery, had happy red lighted numbers, and had lots of blue buttons with serious mathematical symbols on them.

Of course, I err in speaking of this machine in the past tense; it still lives in my desk at home. I can't bear to throw it away. A few weeks ago my handy little calculator was the butt of several jokes from the very modern *DDJ* editors, and this caused me to pause and reflect a bit as we face a new year full of exciting technological advancements.

Motorola had a big shindig here in October to announce its new 68030 chip and gave out solar-powered calculators as a promotional prize. The guys on the staff generously donated the calculator to me so I could finally enter the 1980s. I appreciate this snazzy new tool, but somehow I can't bear to part with my old clunker.

January 1988. We've reached another year closer to 2000, and I find I have a hard time letting some of what made 1987 possible go without mention. Thanks to those great folks on the *DDJ* editorial staff who have gone on to bigger and better (?) things: Nick Turner, Deborah Hart, Vince Leone, and Levi Thomas. Also our columnists, who we hope will still send in an occasional piece of brilliant prose: Michael Ham, Ray Duncan, and Namir Shammas. (Namir is passing the baton of his Structured Programming column into the able hands of Kent Porter as of next issue.)

We have lots of exciting things in store for 1988, the first being the issue you hold in your hands, our annual 680x0 issue. In this issue we debut our new Macintosh column, To the Macs, by Stan Krute. Our editorial schedule for the rest of the year is as follows:

February—Debugging
March—Object-oriented programming
April—AI languages
May—Designing applications
June—Real-time programming
July—Distributed data (hypermedia)
August—Annual C issue
September—Software engineering
October—PostScript; Forth
November—Graphics and video
December—Operating systems

Give Tyler a call with any article ideas.

Starting in February, we will add Examining Room, a series of short product reviews, to *DDJ*'s traditional fare. We won't accept unsolicited reviews but invite you to join the team of "examiners." If you have any ideas, give Ron Copeland a call.

So now we're ready for a new year. I have my new calculator and you have your new magazine. But, being a sentimental sort, I'm determined to hang on to my relic of a calculator. I picture myself well into the 21st century, cuddled around the heat projection unit with my rosy-faced grandkids, all of them eager to hear another tale of days gone by. "Tell us about your first calculator again, Grandma," they will say. "Well," I'll reply, "Back in the late 1970s your Auntie Renèe gave me a pocket calculator for my birthday. . . ."

As we dive headfirst into a year filled with technological promises, let us not forget the people and forces that brought us to where we are today. Best wishes for 1988.

Sara Noah Ruddy

Sara Noah Ruddy
assistant editor

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Editorial
Editor-in-Chief *Michael Swaine*
Editor *Tyler Sperry*
Associate Editor *Ron Copeland*
Assistant Editor *Sara Noah Ruddy*
Technical Editors *Allen Holub*, *Richard Relph*
Contributing Editors *Kent Porter*, *Nāmir Shammas*, *Ernest R. Tello*
Copy Editor *Rhoda Simmons*
Production
Director Art/Production *Larry L. Clay*
Art Director *Michael Hollister*
Assoc. Art Director *Joe Sikoryak*
Technical Illustrator *Barbara Mautz*
Typesetter *Mary E. Lopez*
Cover Photographer *Michael Carr*
Circulation
Circulation Director *Maureen Kaminski*
Fulfillment Coordinator *Francesca Martin*
Book Marketing Mgr. *Jane Sharninghouse*
Subscription Supervisor *Kathleen Shay*
Newsstand Sales Coordinator *Larry Hupman*
Administration
Finance Director *Kate Wheat*
Business Manager *Betty Trickett*
Accounts Payable Supv. *Mayda Lopez-Quintana*
Accts. Receivable Supv. *Laura DiLazzaro*
Advertising
Director *Ferris Ferdon*
Marketing Mgr. *Michael Wiener*
Advertising Coordinator *Patricia Albert*
Account Managers *see page 129*

Publisher
Peter Hutchinson

Dr. Dobb's Journal of Software Tools (USPS 307690) is published monthly by M&T Publishing Inc., 501 Galveston Dr., Redwood City, CA 94063; (415) 366-3600. Second-class postage paid at Redwood City and at additional entry points. *DDJ* is published under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a nonprofit corporation.

Article Submissions: Send manuscripts and disk (with article and listings) to the Associate Editor.

***DDJ* on CompuServe:** Type GO DDJ

Address Correction Request: Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 3713, Escondido, CA 92025. **ISSN 0888-3076**

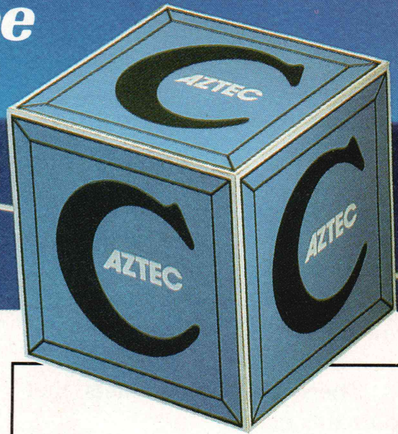
Customer Service: For subscription problems call: outside CA (800) 321-3333; in CA (619) 485-9623 or 566-6947. For book/software order problems call (415) 366-3600.

Subscriptions: $29.97 per 1 year; $56.97 for 2 years. Canada and Mexico add $27 per year airmail or $10 per year surface. All other countries add $27 per year airmail. Foreign subscriptions must be prepaid in U.S. funds drawn on a U.S. bank. For foreign subscriptions, TELEX: 752-351.

Foreign Newsstand Distributor: Worldwide Media Service Inc., 386 Park Ave. South, New York, NY 10016; (212) 686-1520 TELEX 620430 (WUI).

Aztec C

Power to go the distance... Whatever that distance might be



From real time embedded applications to comprehensive commercial applications on Macintosh, IBM PC, Amiga, Atari, and others, Aztec C has earned a well-deserved reputation as an innovative, tough to beat, rock-solid C development system.

But don't just take our word for it—try it yourself. We know that the best way to understand what puts you ahead with Aztec C is to use it. That's why Aztec C systems purchased directly from Manx come with a 30-day, no questions asked, satisfaction guarantee. Call for yours today.

We can also send you information that details the special features and options of Aztec C. Plus information on support software, extended technical support options, and all of the services and specialized support that you may need when you're pushing your software to the limits and . . . beyond.

Aztec C Micro Systems

Aztec C is available for most microcomputers in three configurations: The Professional; The Developer; and The Commercial system. All systems are upgradable.

Aztec C68k/Am .... Amiga
source debugger–optional

Aztec C68k/Mac ... Macintosh
MPW and MAC II support

Aztec C86 .... MS-DOS
source debugger • CP/M libraries

The followng have special pricing and configurations. Call for details.

Aztec C68k/At ..... Atari ST
Aztec C80 .... CP/M-80
Aztec C65 .... Apple II & II GS

Standard System $199
- C compiler
- Macro Assemble
- overlay linker with librarian
- debugger
- UNIX and other libraries
- utilities

Developer System $299
- all Standard System features
- UNIX utilities make, diff, grep
- UNIX vi editor

Commercial System $499
- all Developer features
- source for run time libraries
- one year of updates

MS-DOS Hosted ROM Development Systems

Host + Target: $750 Additional Targets: $500

Targets:
- **6502 family**
- **8080-8085-Z80-Z180-64180**
- **8088-8086-80186-80286/8087-80287**
- **68000-68010-68020/68881**

Components:
- **C compiler for host and target**
- **Assembler for host and target**
- **linker and librarian**
- **Unix utilities make, diff, grep**
- **Unix vi editor**
- **debugger**
- **download support**

Features:
- **Complete development system**
- **Fast development times**
- **Prototype and debug non-specific code under MS-DOS**
- **Compilers produce modifiable assembler output, support inline assembly, and will link with assembly modules**
- **Support for INTEL hex, S record, and other formats**
- **source for UNIX run time library**
- **processor dependent features**
- **source for startup**

Aztec C is available on a thirty-day money back guarantee. Call now and find out why over 50,000 users give Aztec C one of the highest user-satisfaction ratings in the industry.

MANX

C.O.D., VISA, MasterCard, American Express, wire (domestic and international), and terms are available. One and two day delivery available for all domestic and most international destinations.

Manx Software Systems
One Industrial Way
Eatontown, NJ 07724

Call 1-800-221-0440

In NJ or outside the USA, call 201-542-2121
Telex: 4995812 Fax 201-542-8386

CIRCLE NO. 104 ON READER SERVICE CARD

RUNNING LIGHT

Tyler surfaced from writing this month's lead article just long enough to ask me to step onto this page and tell you what to expect from the arrival of HyperCard. Here goes:

1. Expect to see explosive growth in the number of "light" programmers in the Macintosh environment as a result of HyperCard, well beyond the effect of Turbo Pascal when it burst upon the PC environment. There are several facts that support this claim:

It's bundled. A million people will have HyperCard by the end of this year. Nearly all will actually use it (as an application).

The path from using HyperCard to writing your own programs in HyperTalk is smooth. The stages, from browsing to cut-and-paste application building to modifying existing scripts to writing short scripts of your own to developing full stackware applications with HyperTalk, may be the easiest gradient up to programming ever.

The pressure for a truly easy Mac development tool has been building for four years, and the floodgates are now open.

2. Expect prolific output from these "light" programmers. Again, there are several reasons to believe this:

Ease of programming. Even though it embodies principles unfamiliar to a BASIC programmer, HyperTalk is about as easy as BASIC or Pascal.

The power of the language. HyperTalk objects and messages are generally higher-level components than BASIC statements, so you can do more with fewer of them.

Expected aids to programming. Apple is developing enhancements that will make it easier to do more, including toolkits for controlling serial communications, AppleTalk communications, and interactive video. And there will be more public-domain and shareware programming aids like Andy Hertzfeld's PICT file importer. Commercial products will serve as lessons in programming, since the source is examinable.

3. Deduce from these expectations an overwhelming outpouring of stackware (as it is called). The evidence is already coming in: ten megabytes of shareware and public domain stackware was developed in the first three months after the announcement.

4. Expect the inevitable result: HyperGlut. While some amateurs will produce software that is far from amateurish, we will soon see unprecedented amounts of poor work. But obviously-bad software is just noise in the channel; the real HyperGlut will be in the stackware products that meet needs cheaply but poorly. That will, sadly, be used.

5. Expect the end of the predictable Macintosh user interface. That only worked while nearly everybody conformed nearly all the time.

6. Expect bad practices and poor programming style to become ingrained. This is a more depressing thought the more you dwell on it, because there are more ways to program badly with HyperCard than with BASIC. With HyperCard, snippets of code can reside in various places, such as attached to a button or to the card on which the button resides, and the inheritance structure of HyperCard allows events to drop through various objects, searching for a handler.

The object-oriented equivalent of spaghetti: what a concept.

Michael Swaine
editor-in-chief

ARCHIVES

Reaffirmation

"We who have had some degree of involvement with *DDJ* and People's Computer Company (PCC) modestly think of this publication as the lever which, with the slightest degree of pressure, just might move the world a bit. By serving the high end of the technological spectrum, some of our efforts do find their way by mysterious means into products and services which help people. This happens when one of our readers who sees the true potential of computers, and some piece of software we publish, puts them together in new ways." *Editorial, Marlin Ouverson, Editor,* DDJ, *December 1982.*

Tales of Future Passed

"The Xanadu Hypertext System is one of the most powerful systems in existence for managing text in a micro-to-mainframe database environment. It can store arbitrary numbers of documents and retrieve them on demand. It can organize them by using a highly generalized link facility and by allowing (and keeping track of) multiple users on the same system, and allow those users to work with each other on a common document base. It can grow indefinitely over a large distributed network with minimal degradation in performance. . ." Roger Gregory, *"Xanadu - Hypertext from the Future,"* DDJ, *January 1983.*

Pretty Is As Pretty Does

"The laws of entropy insure that the line numbers of a debugged and operational BASIC program give the appearance of having been selected by a KENO machine. In fact, while several texts detail how the boundary conditions of a KENO game lead to predictable outcomes, finished programs seldom exhibit this property. Many a time I have spent an extra hour retyping a finished program while spacing the line numbers evenly just to make it look good." *"Renumbering & Appending BASIC Programs on the Apple II Computer,"* Steve Wozniak, DDJ, *February 1978.*

Dr. Dobb's Journal of Computer Calisthenics & Orthodontia
Running Light Without Overbyte

LETTERS

Optimum Performance?

Dear *DDJ*,

Either Richard Relph's article on optimizing compilers for C (August 1987) has a slight error, or I'm going to have a lot of trouble with my code if I start to use one. I believe the example of common subexpression elimination he refers to as block range optimization is at least module range and could be program range.

Under the K & R definition of C, a two-dimensional array is not necessarily a continuous block of storage. An expression such as *a[i][j]* refers to the *j*th element of the block pointed to by *a[i]*. The element *a* is an array of pointers to appropriate objects. This permits arrays in which rows can have different sizes and in which rows can be switched by swapping pointers and permits arrays to be built up dynamically during execution. Consequently, the identification of *d[i][j]* with *d[0][t0]* where $t0 = i*10 + j$ is not always correct.

Even if we can be assured that the dimensions of *d* will be 10×10 when *copy( )* is called, if the storage of *d* was allocated one row at a time (as is often the case in my programs), there is no assurance that *d[i][j]* can be addressed as in the example. There is even no assurance that the address of *d[i][j]* will bear the same relationship to that of *d[0][0]* as *s[i][j]* will to *s[0][0]*. In the particular example given, that assurance is provided by the declarations of *d* and *s*. (Even that assurance could be compiler implementation dependent, though I know of no compilers for which it would not be correct.) But these were globals, declared outside the function, so the optimization indicated requires at least module-level information and in other cases might require program-level information.

Any optimizing compiler that streamlines multidimensional array addressing in this way will either have to incorporate information beyond the function being optimized or it will have to restrict this technique to certain arrays declared within the function.

Thanks for an interesting article. Keep up the good work.

Clyde Schechter
116 Pinehurst Ave., Apt. D-63
New York, NY 10033

Better (or Worse) Standard

Dear *DDJ*,

I had mixed emotions after reading Richard Relph's description of the forthcoming ANSI C standard in the August 1987 issue. As a programmer working almost exclusively in C, I like the idea of replacing local dialects with a common language, and I like many of the proposed enhancements. I am disturbed, however, by the move to merge the library routines into the definition of the language. At present, the C language is unique in not including library routines as part of the specification, which I think is the right approach.

For example, the article mentions that library function names and macros are to be reserved words, to eliminate the possibility of programmers substituting their own functions for standard ones. I have often substituted such routines to good effect while debugging a program, however. Similarly, it can be useful to set breakpoints on routines such as *strcpy( )*, which will be impossible if the compiler is free to substitute in-line code for my subroutine call.

Although I agree that the library functions should be standardized, making them part of the syntax itself hurts the language. The justification for this seems to be that it makes it easier to optimize the code, but it is a poor trade to take away capabilities in order to go easy on the compiler!

*"Good Grief—now **They've** penetrated **Our** networks!"*

At present, I know what's going on when I call *strcpy( )* in any compiler, and if I want to speed up my program, I can write in-line code myself. With ANSI C, programmers won't know how to write efficient code without knowing which compiler will be used to compile it. This improves portability? Perhaps the ANSI committee needed a few more members who buy compilers along with those who write them.

William F. Linke
286 Dunhams Corner Rd.
East Brunswick, NJ 08816

Richard Relph replies:

Mr. Linke's letter makes a key observation. The C library is now "standardized." Without this change, program portabil-

(continued from page 10)

ity is unobtainable. It should be noted, however, that library routines are not mandated for freestanding environments, such as embedded systems.

I have some difficulty with his final paragraph. He seems to be wanting both maximal speed and maximal portability, a desirable goal, I agree. The two are often at odds with one another, however. Perhaps a solid example would help here. Suppose I need to perform a string copy operation. I could either call *strcpy* or code it in-line. Mr. Linke's assumption is that, by coding it inline, he would get the best performance. This is not necessarily true, even with today's non-ANSI compilers. Most libraries implement *strcpy* in assembly language, using instructions that most C compilers cannot normally generate, such as *REP MOVSB* in the 8086. Therefore, the programmer already doesn't know what's most efficient. Mr. Linke further assumes that he "knows what's going on when I call *strcpy()* in any compiler." Is he aware that some existing compilers perform in-line expansion of key library functions? MetaWare and Microsoft's both do this, and I'm sure others will follow.

By reserving the library function names for the compiler, the compiler can do several things that would otherwise be impossible. First, the library itself can call library routines. If users were allowed to replace *strcpy*, for example, other library routines such as *printf* could not call it. This essentially means that for the normal case when programmers do not replace *strcpy*, but do call it, the library must include two copies, one with some strange name for the library itself and one for the user to call.

Second, the library can be granularized at other than the function level. Although it is nice to have a library that is function granular, some machines cannot support libraries with so many "chunks." If the implementor thought that *strcpy* and *memcpy* made a reasonable pair and put them in a single module, and the programmer replaced *strcpy* but called *memcpy*, what could the linker do?

Last, the compiler can now replace some calls to functions with in-line assembly-language code, taking into account the actual parameters. This is optimal efficiency, allowing the insertion of the assembly-language version of *strcpy* in the code, saving the call and return overhead.

At the risk of being taken literally and in the extreme, let me say that leaving operations at as high a level as possible is always desirable. If I can call *strcpy*, or write it in-line, I will pick the call—for efficiency, for maintenance, for clarity, and for standardization.

Finally, let me say that these compiler features are what people will probably start basing their buying decisions on. This is a net plus. People who want what Mr. Linke wants will probably be able to get it and people who don't will be able to get something else.

```
254 10          UNITS INCHES
254 12 *   10   UNITS FEET
254 36 *   10   UNITS YARDS
10  1           UNITS CENTIMETERS
1000  1         UNITS METERS

\ Usage:
10 FEET    . <cr>  3048  ok
3 METERS   . <cr>  3000  ok
\ ........................
\ etc.
```

Example 1: *Conversion program using the defining word* UNITS

```
VARIABLE  <AS>      0 <AS> !
: AS  -1 <AS> ! ;
: UNITS  CREATE  SWAP , ,  DOES>  D@  <AS> @
       IF  SWAP  THEN  */  0 <AS> ! ;
BEHEAD' <AS>     \ TO MAKE IT LOCAL FOR SECURITY

\ UNIT DEFINITIONS REMAIN THE SAME.
\ Usage:
10 FEET . <cr>  3048  ok
3048 AS FEET .  <cr>  10 ok
```

Example 2: *Code to convert back to input units when outputting*

Dimensional Data Types in Forth

Dear *DDJ*,

This letter is an addendum to Eric Lundquist's comment (Letters, August 1987) on Do-While Jones' article "Dimensional Data Types" (in Ada). His astonishment at the lengthy code required to implement this simple idea in Ada is, of course, justified. As a theoretical physicist and accident consultant who uses dimensioned data every day of my life, I was equally astounded.

Experienced Forth programmers, on the other hand, must have smiled indulgently, murmured "What fools these mortals be!", and passed on. As a fairly recent convert to Forth (and a heavy user of this language for scientific computing), however, I cannot pass up this opportunity (we recent converts tend to be zealots) to exhibit how easily dimensioned data can be implemented in Forth. Perhaps Mr. Lundquist—who feelingly alludes to the advantages of assembly language—will then be sufficiently piqued to give Forth a try.

Suppose you need to input distances in various units. For example, U.S., English, and Continental police reporting accidents might wish to use inches, feet, and yards, or centimeters and meters. Rather than writing different versions of a program, and making users worry about which one they are using, it is simpler to write one program and make unit conversions part of the grammar. Thus, for example, you might keep all internal lengths in millimeters and convert as follows:

```
: INCHES 254 10 */ ;
: FEET [ 254 12 * ] LITERAL 10 */ ;
: YARDS [ 254 36 * ] LITERAL 10 */ ;
: CENTIMETERS 10 * ;
: METERS 1000 * ;
```

The usage would be:

```
10 FEET . <cr> 3048 ok
```

These are more definitions than necessary, of course. A better alternative

(continued on page 126)

EVEN MORE POWER AND FLEXIBILITY

BRIEF 2.0

Users and industry press alike have unanimously proclaimed BRIEF as the best program editor available today. Now, the best gets better, with the release of BRIEF 2.0.

Straight from the box, BRIEF offers an exceptional range of features. Many users find that BRIEF is the only editor they'll ever need, with features like real, multi-level Undo, flexible windowing and unlimited file size. But BRIEF has tremendous hidden power in its exclusive macro language. With it, you can turn BRIEF into your own custom editor containing the commands and features you desire. It's fast and easy.

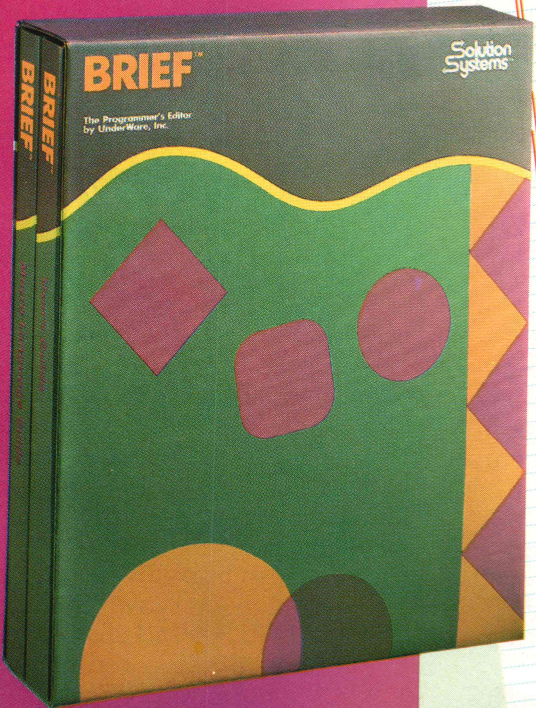
Jerry Pournelle, columnist for BYTE magazine summed it all up by saying BRIEF is, "**Recommended. If you need a general purpose PC programming editor, look no further.**" His point of view has been affirmed by rave reviews in C JOURNAL, COMPUTER LANGUAGE, DR. DOBB'S JOURNAL, DATA BASED ADVISOR, INFOWORLD AND PC MAGAZINE.

One user stated "BRIEF is one of the few pieces of software that I would dare call a masterpiece." Order BRIEF now and find out why. **BRIEF 2.0 is just $195. If you already own BRIEF, call for upgrade information.**

TO ORDER CALL: 1-800-821-2492 (in MA call 617-337-6963)

As always, BRIEF comes with a 30 day money-back satisfaction guarantee.

Solution Systems™
541 Main Street
Suite 410D
So. Weymouth, MA 02190
(617) 337-6963



Look at these BRIEF 2.0 enhancements!

Main Features:

- All new documentation with tutorials on basic editing, regular expressions **and** the BRIEF Macro Language.
- Setup program for easy installation and configuration. (Requires no knowledge of the macro language)
- Increased speed for sophisticated operations like Undo and Regular Expression Search.
- Expanded regular expressions, with matching over line boundaries.
- More block types, with marking by character, line or column.
- Command line editing (move cursor, add and delete characters, specify command parameters).
- Support for more programming languages.
- Optional borderless windows.
- Enhanced large display support, including wider displays.
- Reconfigurable indenting for C files (supports most indenting styles).

Plus the basic features that made BRIEF **SO** popular!

Basic Features:

- Full multi-level Undo
- Windows
- Edit many files at once
- File size limited only by disk space
- Automatic language sensitive indentation

Requires an IBM PC or compatible with at least 192K RAM.

CIRCLE NO. 106 ON READER SERVICE CARD

Introducing the two on earth

The new COMPAQ DESKPRO 386/20™

The world now has two new benchmarks from the leader in high-performance personal computing. The new 20-MHz COMPAQ DESKPRO 386/20 and the 20-lb., 20-MHz COMPAQ PORTABLE 386 deliver system performance that can rival minicomputers'. Plus they introduce advanced capabilities without sacrificing compatibility with the software and hardware you already own.

Both employ an industry-standard Intel® 80386 microprocessor and sophisticated 32-bit architecture. Our newest portable is up to 25% faster and our desktop is actually up to 50% faster than 16-MHz 386 PC's. But we did much more than simply increase the clock speed.

For instance, the COMPAQ DESKPRO 386/20 uses a cache memory controller. It complements the speed of the microprocessor, providing an increase in system performance up to 25% over other 20-MHz 386 PC's. It's also the first PC to offer an optional Weitek™ Coprocessor Board, which can give it the performance of a dedicated engineering workstation at a fraction of the cost.

They both provide the most storage and memory within their classes. Up to 300 MB of storage in our latest desktop and up to 100 MB in our new portable.

It simply works better.

most powerful PC's and off.

and the new 20-MHz COMPAQ PORTABLE 386™

Both use disk caching to inject more speed into disk-intensive applications and both will run MS® OS/2.

As for memory, get up to 16 MB of high-speed 32-bit RAM with the COMPAQ DESKPRO 386/20 and up to 10 MB with the COMPAQ PORTABLE 386. Both computers feature the COMPAQ® Expanded Memory Manager, which supports the Lotus®/Intel®/Microsoft® Expanded Memory Specification to break the 640-Kbyte barrier imposed by DOS.

With these new computers plus the original COMPAQ DESKPRO 386™, we now offer the broadest line of high-performance 386 solutions. They all let you run software being written to take advantage of 386 technology, including Microsoft® Windows/386 Presentation Manager. It provides multitasking capabilities with today's DOS applications to make you considerably more productive. But that's just the beginning. For more information, call 1-800-231-0900, Operator 43. In Canada, call 416-733-7876, Operator 43.

COMPAQ®

386 vs. 030: The Crowded Fast Lane

Picking the fastest CPU can be difficult when not even the benchmarks agree

by Tyler Sperry

Past issues of *DDJ* have detailed the introduction and rise of Intel's 80386 microprocessor and its related software in some detail. The ability of 386 machines to maintain compatibility with MS-DOS software while also giving substantially faster performance is something we all can respect. The 80386 isn't the only CPU that can lay claim to the title of fastest microprocessor, however. Recently, Motorola introduced the latest member of its 68000 line—the 68030—with performance claims that would leave the 80386 in the dust. Obviously, this claim bears some investigation.

The 68000 Gets Respectable

Like many of you, my first experience with a 68000-based computer was the original 128K Macintosh. *Disillusionment* is the mildest term I'd use for my first encounter. Despite the promise of a superfast processor with lots of 32-bit general-purpose registers and plenty of memory (128K!), I was able to go back to my CP/M machine without regret. By burdening the CPU with updating the video and emulating a disk controller, the Mac seemed a perfect demonstration of Grosch's Law[1]; the CPU might be inherently faster than 8-bit machines, but you'd never be able to prove it by the performance.

In the last few years we've seen the introduction of a number of machines that have delivered on the promise of the 68000. The Mac line has matured to produce the 68020-equipped Macintosh II that rivals the performance of the IBM PC AT. At the other end of the spectrum, Sun Microsystems has had enormous success with Unix boxes based on the 68000 and its descendants. Indeed, once you subtract a few proprietary CPUs (from IBM and DEC), workstations are powered almost exclusively by the 68000.

When not programming or reviewing hardware and software, Tyler Sperry works at his day job as Editor of DDJ.

The Paradox of the Installed Software

Fine, you say, but what difference does that make when there are more than 8 million DOS machines out there? And what about the 80386? Doesn't it blow away the 68020?

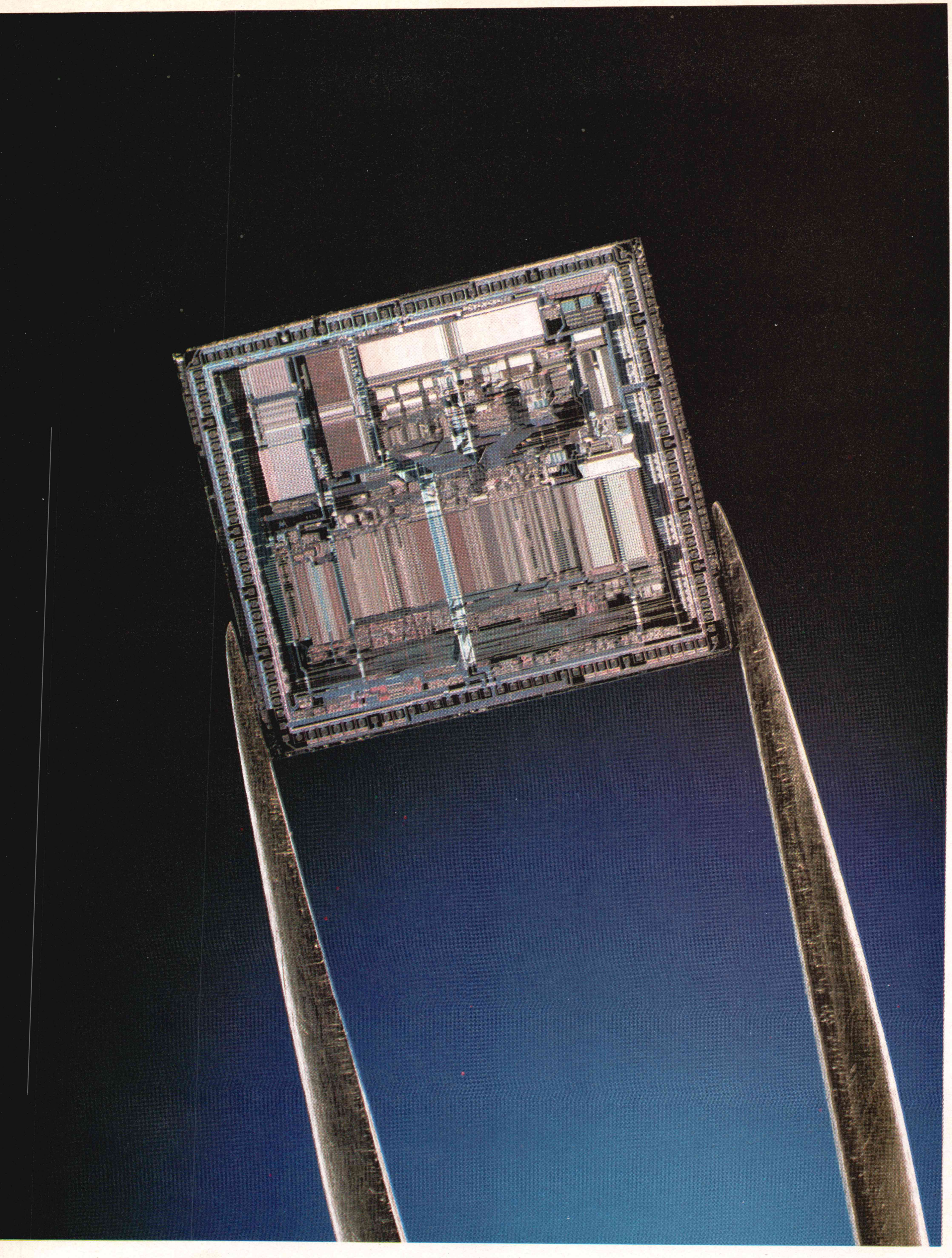
On the first point, I am happy to say that this article is concerned with comparing the latest offerings of the Intel and Motorola lines, not with software and hardware marketing. Still, those questions often come up in a discussion of the technical merits of competing CPUs. (The AMD 29000 looks to be a fantastic chip, but when will you get to program one?) For now, let it suffice to say that:

- Any machine hoping to become the new standard in the personal computer marketplace will have to address the huge software "inertia" of the millions of DOS machines and their software compatibility demands.
- The high-performance CPUs coming out in the next few years will probably make this point moot by providing PC emulation at speeds meeting or exceeding an AT. (This software already exists for Unix boxes.)
- Even if software emulation isn't suitable, 286 "clone cards" are becoming increasingly easier to come by on a variety of buses.

Apples and Oranges

At first glance, it might seem simple to compare the performance of the 80386 and the 68030. Just set up a test jig with some memory and the two CPUs and run some benchmarks. Although that approach might have worked with 8-bit CPUs (the infamous "good old days"), there are a few problems when you get to the new 32-bit processors.

There's memory, for example. Do you furnish the processors with the fastest static RAM available and let them run flat out, or do you run with a "typical" system



(continued from page 17)

(dynamic RAM) and slow the CPUs down with wait states? Arguments can be made on either side.

In the case of the 68030/80386 controversy, this concern has actually been addressed by the chip manufacturers. Both CPUs have provisions for handling the common problems associated with using dynamic RAM slower than the CPU can handle. Both chips have a provision for "burst mode" access, for example, which allows contiguous bytes of memory to be accessed without the delays normally associated with address setup and decoding. In some respects these two CPUs are more similar than different. There is one significant difference in their approach to handling memory access, however, that has a substantial impact on performance.

Cache As Cache Can

Let's take a brief detour down memory lane (so to speak). Back in the days of the 6502 and 8080, access to memory was slow but relatively straightforward. If the processor wanted an instruction, it went out to the memory bus and fetched one.

This procedure began to change with the introduction of the Intel 8088. One of the features of the 8088 was a 4-byte prefetch instruction queue that attempted to separate memory bus activity from computation time. Program instructions were moved from memory into a prefetch queue and then acted upon. Although this sped things up a bit, it was of limited usefulness. (See this month's Letters for more on the subject.) Indeed, the less charitable have referred to the prefetch queue as the prefetch bottleneck.

Eager to please, the engineers at Intel improved things in subsequent Intel designs: the 80386 has a 16-byte prefetch instruction queue. (A simplified schematic is shown in Figure 1a.)

Motorola's attempts at speeding things up became noteworthy with the introduction of the 68020. The 68020 does not have an instruction queue but rather a 256-byte instruction cache. Once an instruction is loaded into the cache from memory, it need not be reloaded unless it's been replaced by a more frequently used instruction. Thus, a small, tight loop can run entirely from on-board cache memory and result in much faster performance. An instruction queue, on the other hand, is by definition limited to operating as an instruction pipeline; any branch taken forces the reloading of the queue.

As you might expect, the addition of an instruction queue can substantially improve a processor's performance. The amount of improvement will, of course, depend on how many tight loops there in your code. (Yet another reason to be wary of small benchmarks.) Thayne Cooper and some engineers at Sperry ran both the 68020 and 80386 through some modified *EDN* benchmarks and published the results in *IEEE Micro*.[2] While a 16-MHz 80386 was able to surpass a 16-MHz 68020 with a disabled cache, enabling the cache better than halved the original 68020 benchmark times. (The cached 68020 beat the 80386 in all tests except the string search benchmark.)

Lies, Damn Lies, and Benchmarks

Now, given those benchmarks results, it'd seem pretty clear cut. The performance improvement of boosting the clock speed to 20 MHz should be pretty much the same for either chip. Score them neck and neck—with the edge to the 68020—and we're done, right?

Alas, as my friend Jerry Pournelle would say, it isn't all that simple. The benchmarks done by Cooper were modified 16-bit *EDN* benchmarks, performed on special hardware. The hardware was designed to keep things as

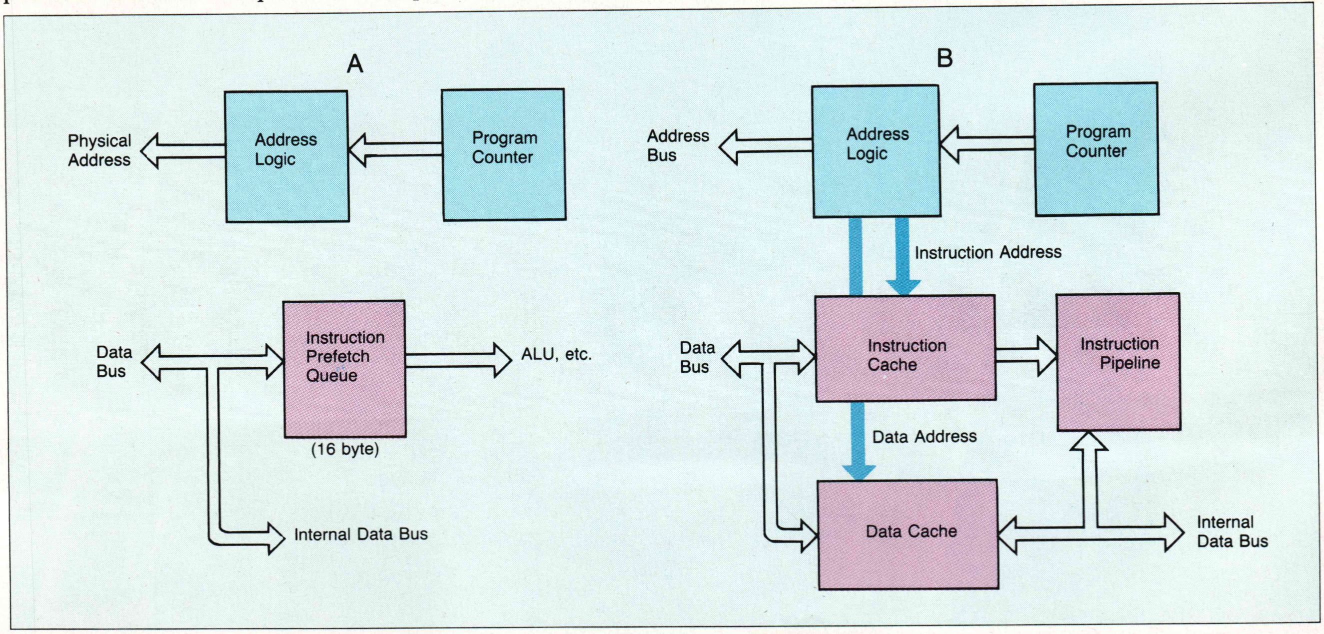


Figure 1: *Simplified view of the memory interface for the 80386 and 68030. The 80386 (a) has a 16-byte prefetch instruction queue. The 68030 (b) features a modified Harvard architecture with separate 256-byte caches for both instructions and data.*

(continued from page 18)

equal as possible for the various processors (the 32032, 32100, and the 80286 were also tested).

Unfortunately, life isn't always fair: your choice of machines will often not include units comparable in all aspects except CPU; sometimes the benchmarks used in a test don't always bear a strong resemblance to your actual application and environment; and the compiler used can impact your results tremendously.

Consider the case of poor Richard Grehan at *Byte*.[3] He took several varieties of 386 computers and accelerator cards, compiled some benchmarks, and ran them. Then he did the same thing for a Mac II and some 68020 accelerator cards in different environments. If you have some experience with benchmarks (or if you read *Byte* regularly), you can anticipate what he found: the 80386 outperformed the 68020 in the majority of tests.

How to explain this? Well, there are some things to note in the *Byte* article benchmarks. First of all, these tests were performed with the intent to test mathematical performance. The only nonmathematical tests were the infamous Sieve and a quicksort routine. As the commercials say, your actual milage may vary.

Second, although Grehan tried to use the same compiler vendor for all machines, this wasn't always feasible. Some of the compilers used for two of the 68020 machines gave substantially better times than the Macintosh compiler used for the other tests, and in fact these times were in the same neighborhood as the best 80386 time (a 16-MHz Compaq 386 with an 80387 coprocessor, in case you were wondering).

The lesson here is unfortunately all too clear. The best benchmark is your target application, ported to the prospective machine. Depending on the optimizations offered by the compiler and individual machine peculiarities, you'll find benchmarks vary widely—there are too many confounding variables for a categorical statement that one chip is better than another. Still. . . .

Going Back to School

After all that discussion and equivocation on the subject of the 68020 vs. the 80386, you'd expect making a clear statement on the relative performance of the 68030 wouldn't be too plausible. After all, as of this writing, there aren't many 68030 machines available to test. (Both Apple and NeXT are rumored to be working on 68030 designs; both refuse to comment on unannounced products.) In reading through the literature, though, I came across some things that can let us make a pretty good guess.

To start with, the 68030 implements a modified Harvard architecture along with expanded caching. A Harvard architecture machine uses separate address and data buses for both instructions and data; in the 68030, a modified Harvard scheme is employed, in which separate buses are used internally and then multiplexed for access to the system. Figure 1b, page 18, shows a simplified schematic of the 68030's memory interface.

386 vs. 030

(continued from page 20)

Notice that there are now two 256-byte caches: one for instructions and one for data. Given the radical improvement a cache made in the 68020's performance, you can see why Motorola is proudly trumpeting the 68030 as "twice the microprocessor." Of course, it didn't hurt that the chip runs at a clock speed of 25 MHz.

Conclusions

Given that the 80486 is still quite a ways away, Intel would probably like you to believe that a 16-MHz 80386 is equal to a 20-MHz 68020 and that its 20-MHz 80386 is equal (or better) to a 25-MHz 68030. Motorola, as you might expect, has a different view: a fast 68020 is a match for any 80386 and the 68030 blows away an 80386 at any speed.

Aside from the engineer's instinctive distrust of (other people's) benchmarks, and despite the vendor charges and countercharges concerning the benchmarks, there are some clear lessons:

- Other things being equal, an 80386 and a 68020 will perform at roughly the same rate: bloody fast.
- A 68030 at 25 MHz will probably be faster than any 80386 you find. How much faster, though, will depend a great deal on your software and compiler.
- If your application is primarily number crunching, a fast math coprocessor is essential and its presence or absence will probably swamp other aspects.
- A weak compiler can mislead you on the performance of a given system. Conversely, a highly optimizing compiler can completely destroy the value of a poorly constructed benchmark.[4]
- Beware of virtual machines. Today's 5-MHz PC clone is faster than a 50-MHz 80486 box that won't be shipping for another six months.

Notes

1. Ted Nelson, *Computer Lib/Dream Machines* (Microsoft Press, 1987).
2. T.C. Cooper, W.D. Bell, et al., "A Benchmark Comparison of 32-Bit Microprocessors" *IEEE Micro*, vol. 6, no. 4 (August 1986).
3. Richard Grehan, "The New Generation: A Closer Look" *Byte* (September 1987).
4. Richard Relph, "Optimizing Compilers for C" *DDJ*, vol. 12, no. 8 (August 1987).

Bibliography

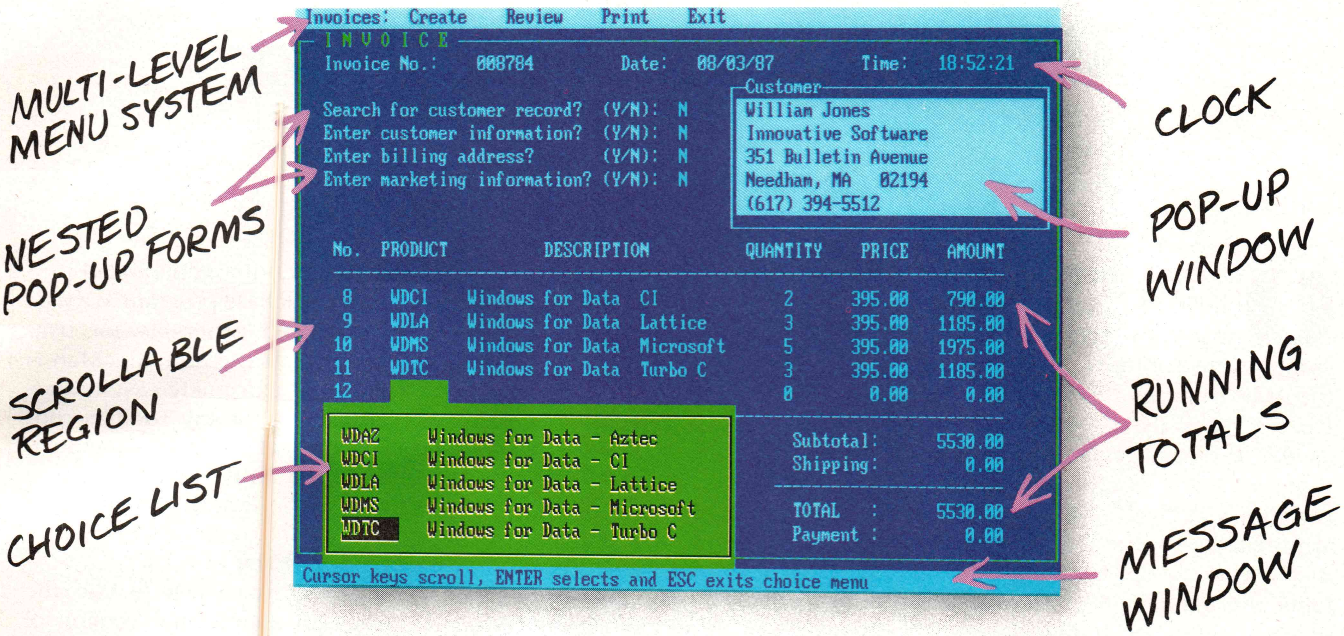
Motorola. *MC68030 User's Manual*. Motorola, 1987.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 6.**

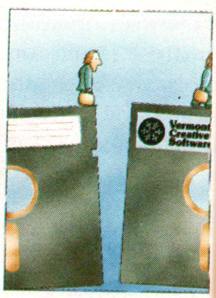
WINDOWS FOR DATA®

Uncommon Screens



If you program in C, take a few moments to learn how Windows for Data can help you build a state-of-the-art user interface.

- ☐ Create and manage menus, data-entry forms, context-sensitive help, and text displays — all within windows.
- ☐ Provide a common user interface for programs that must run on different machines and operating systems.
- ☐ Build a better front end for any DBMS that has a C-language interface (most popular ones do).



FROM END TO BEGINNING

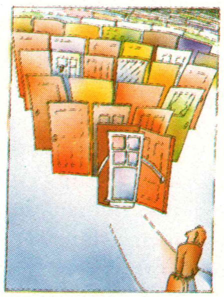
Windows for Data begins where other screen packages end, with special features like nested pop-up forms and menus, field entry from lists of choices, scrollable regions for the entry of variable numbers of line items, and an exclusive built-in debugging system.

NO WALLS

If you've been frustrated by the limitations of other screen utilities, don't be discouraged. You won't run into walls with Windows for Data. Our customers repeatedly tell us how they've used our system in ways we never imagined — but which we anticipated by designing Windows for Data for unprecedented adapatability. You will be amazed at what you can do with Windows for Data.

YOU ARE ALWAYS IN CHARGE

Control functions that you write and attach to fields and/or keys can read, compare, validate, and change the data values in all fields of the form. Upon entry or exit from any field, control functions can call up subsidiary forms and menus, change the active field, exit or abort the form, perform almost any task you can imagine.



OUR WINDOWS WILL OPEN DOORS

Our windows will open doors to new markets for your software. High-performance, source-code-compatible versions of Windows for Data are available for **PCDOS** (**OS/2** soon), **XENIX, UNIX,** and **VMS.** PCDOS versions are fully compatible with Microsoft Windows, TopView, and DESQview. **No royalties.**

You owe it to yourself and your programs to try Windows for Data. If not satisfied, return for a full refund. Call for **FREE DEMO.**

Vermont Creative Software

21 Elm Ave., Richford, VT 05476
Telex: 510-601-4160 VCSOFT FAX 802-848-3502
Tel.: 802-848-7731 ext. 31
Prices: PCDOS* $395; XENIX, VMS, UNIX Call.

CIRCLE NO. 111 ON READER SERVICE CARD

A Programmers' Database for the Macintosh Toolbox

by Abdullah Al-Dhelaan and Ted G. Lewis

Putting Inside Macintosh inside your Macs

One challenge to writing application software for Apple's Macintosh is the complex environment. Programming the Mac requires the use of an extensive set of ROM routines known as the Toolbox. These routines, although largely responsible for the machine's radical ease of use, add appreciably to the Mac programmer's work load. The Toolbox contains more than 600 routines, and many of them are required in writing even the smallest application. Most programmers find themselves continually referencing Apple's massive documentation, *Inside Macintosh*, for technical details on the numerous procedures, functions, and data structures of the Toolbox.

At first, using *Inside Macintosh* as a handy reference manual might seem a minor chore. It has a good index and is divided into five logical volumes. Unfortunately, information retrieval time becomes a dominant factor in the development process, and it quickly becomes obvious that an on-line, electronic means of retrieval is needed.

Our program, MacMan, is an on-line programmers' database that contains much of the key information found in *Inside Macintosh*. Programmers can use MacMan to fetch the name, parameters, comments, and often-needed descriptions of the Toolbox routines. This information can be accessed from within an editor as an aid to documentation or can be used simply as a way to acquire a better understanding of the inner workings of the Mac. Currently, the database contains more than 500K of text describing the Toolbox routines as well as many other useful facts about the Macintosh.

Abdullah Al-Dhelaan and Ted Lewis work in the Computer Science Dept. Oregon State University, Corvallis, OR. 97331.

The design goals for MacMan were simple: the program should be convenient to use, and it should contain useful information. Above all else, we knew that it had to be convenient to be compelling. Convenient meant both simple and fast. We knew that programmers wouldn't accept MacMan if they had to learn a new language or come to grips with a complex database system.

We also knew that the information contained had to be accurate and useful. We could have written the information in a more useful form than that of *Inside Macintosh*, and we could have selected information from various other sources. We rejected these alternatives, however, because we didn't want to risk introducing errors or to accidentally include ambiguous passages.

We therefore elected to copy carefully selected passages directly from *Inside Macintosh*. We worked hard to convince Apple Computer to let us use the copyrighted contents of *Inside Macintosh* specifically to avoid errors or misrepresentations of fact.

MacMan is not a generalized programmers' database program. We willingly sacrificed generality for user convenience. As a result, MacMan is both fast and extremely easy to use. The description of any Toolbox routine can be retrieved from the 500K database file and displayed in a window—all in less than a second.

Using MacMan

There are a two ways to use the database: an abbreviated version of MacMan can be installed in the system file as a desk accessory (DA), or a full-fledged application version of the program can be launched from the desktop.

A DA is a special program that can run concurrently in memory with another application. Most *DDJ* readers are familiar with the concept of DAs from TSR (terminate-and-stay-resident) programs such as SideKick, but these products illustrate an ad hoc solution to the problem of writing a DA. In contrast to the PC, the Mac allows DAs to be integrated into the Macintosh operating environment. It's this integration that we'll be emphasizing in this article. We'll describe the MacMan desk accessory and then show how it was implemented using the Macintosh Toolbox functions.

Two simple access methods are provided through the menu: Find by Name and View by Category (see Figure 1, page 25). Find by Name, as you might expect, retrieves the desired Toolbox routine by its name. If you don't know the routine's name, you can select the View by Category menu item (see Figure 2, page 25). View by Category lets you select one of the 28 managers as a category

and then browse through it. When you browse through a category, the Toolbox routine names are displayed in alphabetical order—selecting one of them results in a full display of the routine's description (see Figure 3, below). If MacMan is unable to find a Toolbox routine, it reports the error and asks if it should browse all the routines that begin with the same letter (see Figure 4, page 26).

The subject of this article, DAMacMan, is a version of the database that runs as a desk accessory. Before delving into the inner structure of DAMacMan, however, we'll first review the structure of the typical Macintosh application and how it relates to calling DAs.

The Structure of Mac Applications

Every Macintosh application consists of at least one event loop that determines what operations the application's user is allowed to perform. The event loop must handle all user interactions such as mouse clicks, menu selections, and icon manipulations. The existence of an event loop makes Macintosh applications resemble real-time control programs more than traditional interactive sequential programs.

Every well-behaved application must include a Toolbox call to *SystemTask* so that periodic actions, such as updating the system clock, can be performed by the Macintosh operating system. In Example 1, page 26, we show only one *SystemTask* call for each pass through the main event loop, but in general *SystemTask* should be called at least once every 16 clock ticks (a tick is defined as a 60th of a second). If the application is doing a lot of work on each pass of the event loop, then the *SystemTask* call should be made more often.

The application calls *GetNextEvent* each time through the event loop in order to find out what events have taken place since the previous pass. *GetNextEvent* calls the *SystemEvent* routine, which simply intercepts the stream of events, and if an event belongs to the DA, that event is shuttled to the DA rather than to the application.

Suppose the user selects the DA menu and presses the mouse button; this will cause a *mouse-down* event (*mouseDown*) to be generated by calling the *DoMouseDown* routine. The application programmer must write the *DoMouseDown*

Figure 1: *A Toolbox routine is retrieved by giving either its name or its manager category.*

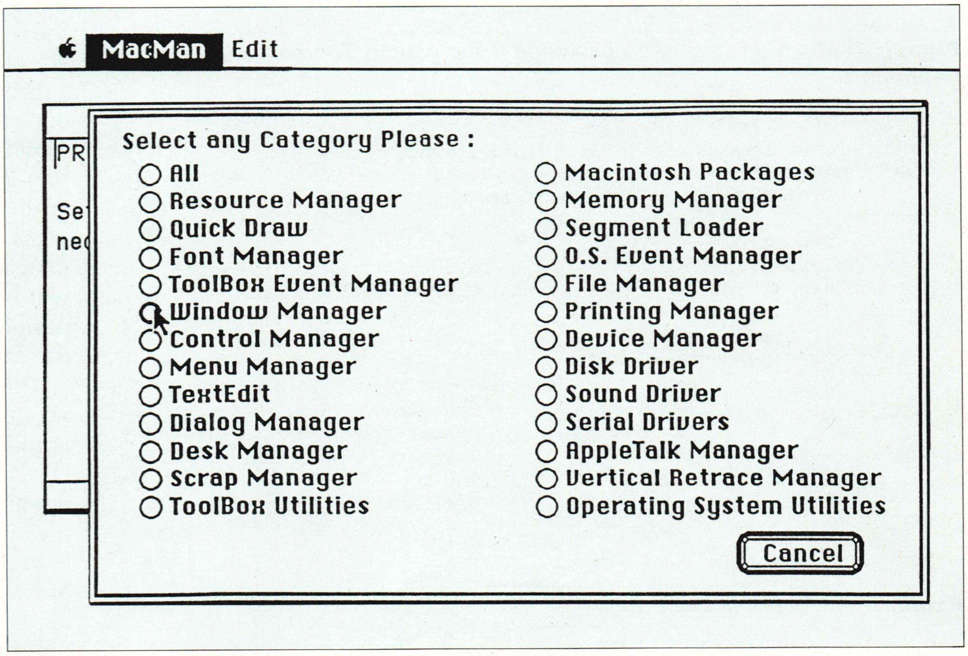


Figure 2: *The MacMan catagories cover the ROM routines for both the user interface Toolbox and operating system.*

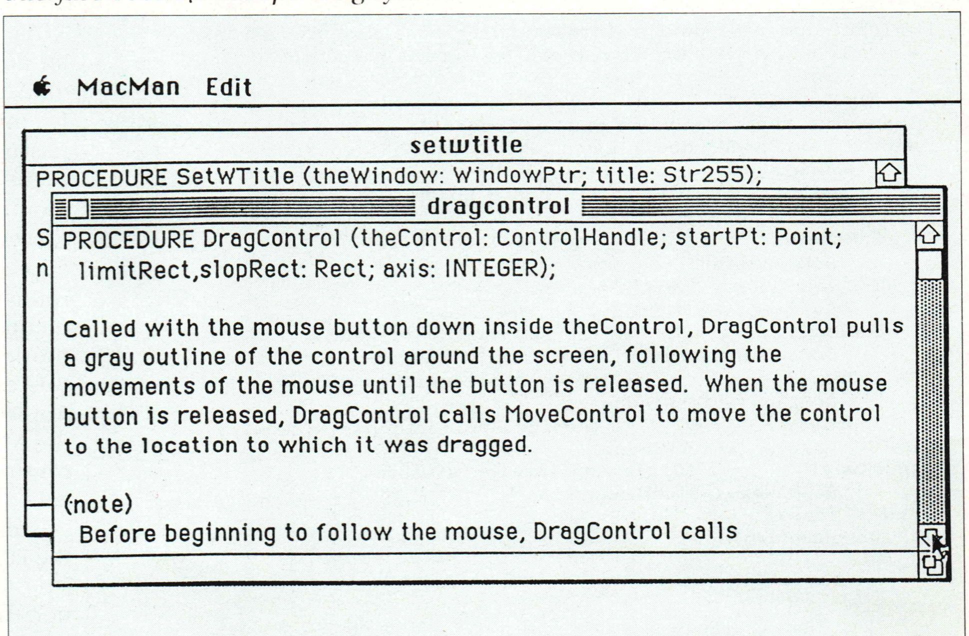


Figure 3: *An example of a displayed Toolbox routine*

Figure 4: *MacMan reverts to browsing if the named Toolbox routine can't be found.*

```
PROCEDURE SimpleEventLoop( Var Event: EventRecord );
   Var UserAction    : Boolean;     {Has the user done something??}
          Finished          : Boolean;     {Exit When User Quits}
   Begin
          Repeat
          SystemTask;                   {To support periodic events}
          UserAction := GetNextEvent( AllEvents, Event);
                                             {To invoke SystemEvent}
          If UserAction then            {Handle the event...}
          Case Event.what Of
                      mouseDown: DoMouseDown(Event, Finished );
                      KeyDown        : DoKeyDown(Event, Finished );
                      ActivateEvt    : DoActivate (Event, Finished );
                      UpDateEvt      : DoUpdate(Event, Finished );
          End;        {Case}
          Until Finished;               {terminate the program?}
End;                                    {SimpleEventLoop}
```

Example 1: *A simple event loop*

```
PROCEDURE DoMouseDown(       Var Event       : EventRecord;
                                    Var Finished: Boolean );
Var
   whichWindow : WindowPtr;{Window that mouse was pressed in}
   whereIs    : INTEGER;       {Part of screen where mouse
                                                  was pressed}
Begin {DoMouseDown}
   {Where on the screen was mouse pressed?}
   whereIs := FindWindow ( Event.where, whichWindow);
   Case whereIs Of
      InDesk:                 {In Empty Space...}
         {Do nothing};
      InMenuBar:              {Menu Selection...}
         DoMenuClick;
      InSysWindow:   {Aha! In a DA... }
         SystemClick ( Event, whichWindow);
      InContent:              {In Application's window...}
         DoContent (whichWindow);
      InDrag:                 {Drag Application's window...}
         DoDrag (whichWindow);
      InGrow:                 {Resize Application's window...}
         DoGrow (whichWindow);
   InGoAway:         {Close Application's window...}
         DoGoAway (whichWindow)
   End ·  {case}
End; {DoMouseDown}
```

Example 2: *The DoMouseDown routine*

MAC DATA BASE

(continued from page 25)

routine in such a way as to call the appropriate DA. The code necessary to do this is shown in Example 2, below.

The events that are diverted from the application to the DA are channeled into the DA processing code in two ways, as shown in the *DoMouseDown* procedure. The first way is through the *SystemClick* Toolbox routine, as shown in case *InSysWindow* of *DoMouseDown*, and the second way is through a menu selection.

When a mouse-down event occurs in a system window, the application code should call *SystemClick*. If the mouse-down event is in a DA window, *SystemClick* takes care of processing the event instead of the application. This case will be discussed later as it is what happens when the DA is already on the screen.

The mouse-down event could also occur in the DA menu (under the apple), which would mean that the DA is to be activated (this is called opening the DA). This is the case we are most interested in for the time being. If the user has selected the MacMan DA, for instance, then the application program must handle the opening of the DA from the *DoMenuClick* routine, shown in Example 3, page 30.

As shown in *DoMenuClick*, when an application calls the *MenuSelect* (or *MenuKey*) Toolbox routine, a call is made to *SystemMenu*, which passes the event to the DA (if the event is a mouse-down in the DA menu). The DA must then handle the event and return a zero to the application. Otherwise, *MenuSelect* returns a long integer containing the menu number in its *HiWord* and the item number in its *LoWord*. In this case, when the user selects the Apple menu, and within this menu, the MacMan DA, the *DoAppleChoice* routine (see Example 4, page 30) is called to activate the DA.

This sequence of actions opens the DA and prepares it for use alongside the currently running application. In summary, the sequence is:

- A mouse-down event occurs and

How to tell the difference between DESQview™ 2.0 and any other environment.

Selecting DESQview, the environment of choice, can give you the productivity and power you crave, without the loss of your old programs and hardware. If you like your existing programs, want to use them together, transfer data between them, print, sort, communicate with or process-in-background, yet still have the need to keep in place your favorite PC(8088, 8086, 80286 or 80386), DESQview is the "proven true" multitasking, multi-windowing environment for you. Best of all, DESQview 2.0 is here now, with all the money saving, time saving, and productivity features that others can only promise for the all-too-distant future.

And with DESQview's new graphics enhancements for Hercules, CGA, EGA, and VGA, Version 2.0 still offers the same award winning and pioneering features for programs that earned DESQview its leadership, only now you can also run desktop publishing programs, CAD programs, even GEM-™, Topview-™, and Microsoft Windows-™ specific programs. In some cases you'll add as little as 10–40K to your system overhead. Now you can have multi-tasking, multi-windowing, break the 640K habit too and still get an auto dialer, macros, menus for DOS and, for advanced users, a new complete application programmer's interface capability. No wonder that over the years, and especially in recent months, DESQview, and now DESQview 2.0 have earned extravagant praise from some of the most respected magazines in the industry.

"Product of the Year" by readers vote in InfoWorld.

"Best PC Environment" by popular vote at Comdex Fall in PC Tech Journal's "System Builder" Contest.

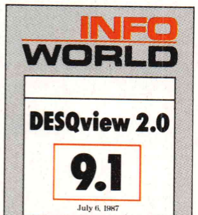
"—I wouldn't want to run an IBM or compatible computer without DESQview"—InfoWorld, Michael Miller.

"A colossus among windowing environments"... "will run almost anything"—PC Week, Marvin Bryan.

"Windows, promises, but DESQview delivers"—MICROTIMES, Birell Walsh.

No other environment has consistently pioneered features, openness, and productivity. See for yourself. Send in the coupon. The possibilities are endless with DESQview 2.0.

One picture is worth a thousand promises.



Attention Programmers: For more information about Quarterdeck's API, and future 386 program extensions, call us today.

SYSTEM REQUIREMENTS
IBM Personal Computer and 100% compatibles (with 8086, 8088, 80286 or 80386 processors) with monochrome or color display; IBM Personal System/2 • Memory: 640K recommended; for DESQview itself 0-145K • Expanded Memory (Optional): expanded memory boards compatible with the Intel AboveBoard; enhanced expanded memory boards compatible with the AST RAMpage • Disk: Two diskette drives or one diskette drive and a hard disk • Graphics Card (Optional): Hercules, IBM Color/Graphics (CGA), IBM Enhanced Graphics (EGA), IBM Personal System/2 Advanced Graphics (VGA) • Mouse (Optional): Mouse Systems, Microsoft and compatibles • Modem for Auto-Dialer (Optional): Hayes or Compatible • Operating System: PC-DOS 2.0-3.3; MS-DOS2.0-3.2 • Software: Most PC-DOS and MS-DOS application programs; programs specific to TopView 1.1, GEM 1.1 and Microsoft Windows 1.03 • Media: DESQview 2.0 is available on either 5¼″ or 3½″ floppy diskettes

Rush me DESQview 2.0! Today!

No. of Copies	Media 3½″/5¼″	Product		Retail Price ea.	Total
		DESQview 2.0		$129.95	$
		Shipping & Handling	USA Outside USA	$ 5.00 $ 10.00	$
			Sales Tax (CA residents)	6.5%	$
Payment: ☐ Visa ☐ MC ☐ AMEX ☐ Check				Amount Enclosed	$

Credit Card: Valid Since ______/______ Expiration ______/______

Card Number: ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Credit Card Name ______________________

Shipping Address ______________________

City ______ State ______ Zip ______ Telephone ______

Mail to: Quarterdeck Office Systems, 150 Pico Boulevard, Santa Monica, CA 90405.
NOTE: If you own DESQview call us for a special upgrade offer, or send in your DESQview registration card. AST Special Edition users included.

DESQ view™
Quarterdeck

PC MAGAZINE EDITOR'S CHOICE NOV. 24, 1987

Quarterdeck Office Systems • 150 Pico Boulevard, Santa Monica, CA 90405 • (213) 392-9851

VEDIT PLUS

#1 PROGRAMMABLE EDITOR

Call 1-800-45-VEDIT for FREE Fully Functional Demo Disk

Stunning speed. Unmatched performance. Total flexibility. Simple and intuitive operation. The newest VEDIT PLUS easily satisfies the most demanding computer professional.

Try a Dazzling Demo Yourself.

The free demo disk is fully functional—you can try all features yourself. Best, the demo includes a dazzling menu-driven tutorial—you experiment in one window while another gives instructions.

The powerful "macro" programming language helps you eliminate repetitive editing tasks. The impressive demo/tutorial is written entirely as a "macro"—it shows that no other editor's "macro" language even comes close. And VEDIT PLUS is only 40K in size.

Go ahead. Call for your free demo today, You'll see why VEDIT PLUS has been the #1 choice of programmers, writers and engineers since 1980.

Only VEDIT PLUS is this Flexible.

The installation lets you pick from closely emulating the keyboard layout of Word Perfect, WordStar and others. Or you can easily create your own layout and even your own editing functions. Supports any screen size—you pick screen colors and attributes.

Supports the IBM PC, XT, AT and PS/2. Also supports MultiLink, PC-MOS/386, Concurrent DOS and most networks. Also available for MS-DOS, FlexOS (protected mode), CP/M-86 and CP/M. (Yes, we support windows on most CRT terminals, including CRTs connected to an IBM PC.) Order direct or from your dealer. $185.

Special: VEDIT (single file, no windows) for CP/M—$49.

- **Fully Network Compatible**
- **Call for XENIX-286 version**
- **30 Day Money-back guarantee**

Compare Features and Speed

	BRIEF	Norton Editor	PMATE	VEDIT PLUS
'Off the cuff' macros	No	No	Yes	Yes
Built-in macros	Yes	No	Yes	Yes
Keystroke macros	Only 1	No	No	Unlimited
Multiple file editing	20 +	2	No	20 +
Windows	20 +	2	No	20 +
Macro execution window	No	No	No	Yes
Pop-up menus	No	No	No	Yes
Execute DOS commands	Yes	Yes	Yes	Yes
Automatic processing of Compiler errors	Yes	No	No	Yes
"Cut and paste" buffers	1	1	1	36
Undo line changes	Yes	No	No	Yes
Paragraph justification	No	No	No	Yes
Convert to/from WordStar	No	No	No	Yes
On-line calculator	No	No	No	Yes
Configurable Keyboard	Hard	No	Hard	Easy
43 line EGA support	Yes	No	No	Yes
Manual size/index	250/No	42/no	469/Yes	380/Yes
Benchmarks in 120K File:				
2000 replacements	1:15 min	34 sec	1:07 min	6 sec
Pattern matching search	20 sec	Cannot	Cannot	2 sec
Pattern matching replace	2:40 min	Cannot	Cannot	11 sec

VEDIT and CompuView are registered trademarks of CompuView Products, Inc. BRIEF is a trademark of UnderWare, Inc. PMATE is a trademark of Phoenix Technologies Ltd. Norton Editor is a trademark of Peter Norton Computing Inc. MultiLink and PC-MOS/386 are trademarks of the The Software Link, Inc. CP/M and FlexOS are trademarks of Digital Research. MS-DOS is a trademark of Microsoft.

*Also available for TI Professional, Tandy 2000, DEC Rainbow, Wyse WY700 and others.
*Demo disk is fully functional, but does not readily write large files.

CompuView

1955 Pauline Blvd., Ann Arbor, MI 48103
(313) 996-1299, TELEX 701821

CIRCLE NO. 113 ON READER SERVICE CARD

MAC DATA BASE

(continued from page 26)

is handled by the event loop.

- The *DoMouseDown* routine decides the event is *InMenuBar*.
- The *DoMenuClick* routine decides the event is an *AppleMenu* event.
- The *DoAppleChoice* routine decides the event is a DA selection.
- The name of the DA (*accName*) is obtained and the DA opened.

The Structure of a Mac DA

A DA is a "mini-application" that can be run concurrently with a Macintosh application. A DA cannot exceed 32K of executable machine code and data and is installed in the start-up disk system file using a Macintosh utility program called the Font/DA Mover. Once a DA is installed, it no longer has a file or an icon visible from the desktop. Instead, the user opens a DA by selecting it from the standard Apple menu, which by convention is the first in the menu bar.

After a DA has been opened, its window, if any, is displayed on the desktop and it becomes the active window. To close a DA, the user can click the DA's close box (in its own title bar), or another program can call the function *CloseDeskAcc* to close it. The DA will then disappear and the frontmost window will become the active window.

A desk accessory may have a menu of its own, which will be added to the menu bar when it is active and deleted when it is not. The *Cut*, *Copy*, and *Paste* commands in a standard Edit menu can be used by an active DA. They are very useful for copying and pasting between the DA and the application or another DA.

Writing a DA

Writing a DA is a lot more difficult than writing a "plain vanilla" application because a DA has no main procedure, no main event loop to obtain events, and no global variables.

Technically, a DA is known as a Macintosh device driver, and each DA is required to have three special procedures: *Open*, *Close*, and *Ctl* (for control). These procedures are called directly by the operating system through a special table called the DA Header.

Each of these procedures requires two formal parameters of type *Device Control Record* and *Parameter Block Record*. A *Device Control Record* is created when a DA is opened and destroyed when it is closed. A *Parameter Block Record* is created by the operating system each time any of the three routines is called. It is used to inform the DA about the purpose of the call.

The Macintosh operating system calls *Open* whenever a DA is selected from the Apple menu and calls *Close* whenever the close box on the title bar of the DA's window is clicked or *CloseDeskAcc* is called by some other program. Between these two calls, the system will call *Ctl*.

What DA Procedures Do

When the *Open* procedure is called to open the DA, it will:

1. Create the DA window, if there is one
2. Set the *WindowKind* field of the window's *WindowRecord* to the DA's driver reference number. (This field is set so the operating system can call the correct DA when an event

MAC DATA BASE
(continued from page 29)

occurs in a DA window.)
3. Set the *DctlWindow* field of the DA's *Device Control Entry Record* to the window pointer.
4. Allocate space for global data in the field *dctlstorage* of the *Device Control Entry Record.*

```
PROCEDURE DoMenuClick;       {  Handle mouse-down event in
                                               menu bar.  }
Var
   menuChoice        : LONGINT;     {Menu ID and item number}
   theMenu    : INTEGER;     {Menu ID of selected menu}
   theItem    : INTEGER;     {Item number of selected item}

Begin {DoMenuClick}

   menuChoice := MenuSelect (TheEvent.where);
      if menuChoice <> 0            {Application, or DA?}
      Then begin                    {Application...}
          theMenu := HiWord(menuChoice);{Get menu ID}
          theItem := LoWord(menuChoice);{Get item number}
          Case theMenu Of
              AppleID:       {Make selection from Apple menu}
                      DoAppleChoice (  GetMenu( AppleID ), theItem);
              FileID:        {Make selection from File menu}
                      DoFileChoice  ( GetMenu( FileID ), theItem);
          EditID:     {Make selection from Edit menu}
                      DoEditChoice  ( GetMenu( EditID ), theItem)
   End;   {case}
   End;   {if}
End;          {DoMenuClick}
```

Example 3: *The DoMenuClick routine*

```
PROCEDURE DoAppleChoice (Var  AppleMenu: MenuHandle;
                                    theItem    : INTEGER);
Var
   accName    : Str255;        {Name of desk accessory}
   accNumber  : INTEGER;       {Reference number of desk accessory}

Begin {DoAppleChoice}
Case theItem of
   AboutItem:        {Application's About... Item}
      DoAbout;
   otherwise {Must be a DA...}
      Begin
      GetItem (AppleMenu, theItem, accName);
                                                {Get accessory name}
      accNumber := OpenDeskAcc (accName)    {Open desk accessory}
      End {otherwise}
 End {case}
End;         {DoAppleChoice}
```

Example 4: *The DoAppleChoice routine*

```
Procedure DoEditChoice (...., theitem : Integer);
  { Handle choice from Edit Menu }
   Begin                    { DoEditChoice}
      if Not SystemEdit (theItem-1)
     Then
         Case theItem of
            cutitem      : DoCut;
            copyitem   :  DoCopy;
            Pasteitem  : DoPaste;
     end; { Case}
End;      {DoEditChoice}
```

Example 5: *The DoEditChoice routine*

5. Initialize the global data.

Close is called to close the DA. It first disposes of its window and stores nil in the *DctlWindow* field of the *Device Control Entry Record*, then it disposes of any global data it might have allocated in the *dctlstorage* field of the *Device Control Entry Record*.

Ctl is called to enable the DA to handle the action indicated by the *csCode* field of the *Parameter Block Record*. There are nine such actions, as shown in Table 1.

Passing Text

An application must have the standard Edit menu if it wants to support passing text to and from DAs. The order of items in this menu is important, but the menu can be made longer by adding items at the end.

A DA program consists of at least three special procedures called Open, Close, and Ctl.

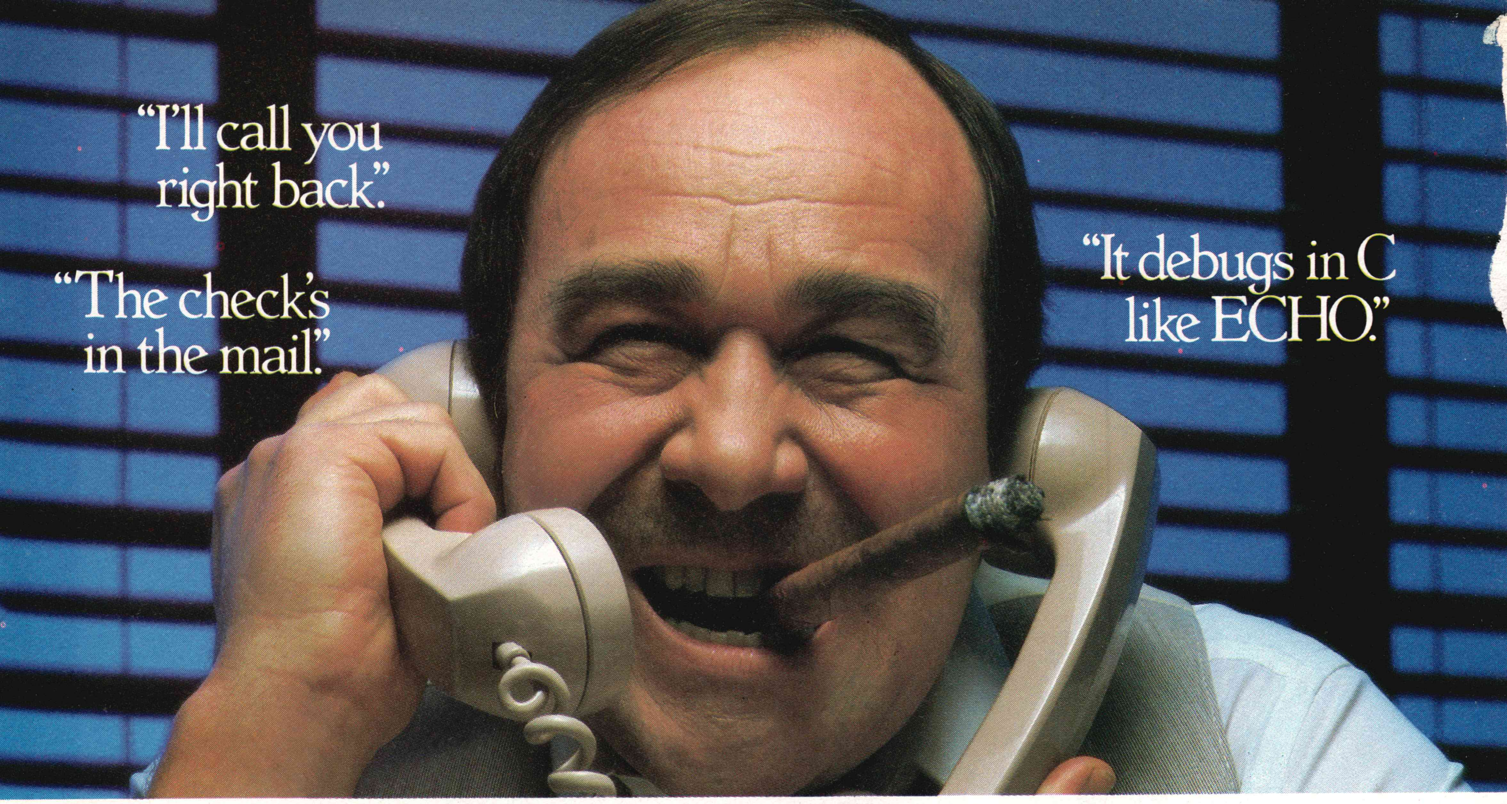
The standard Edit menu contains Undo, Cut, Copy, Paste, and Clear. When a user chooses one of these commands, the application must call Toolbox routine *SystemEdit* from within *DoEditChoice* (see Example 5, page 30). The menu items are numbered 0 through 5 internally, which is why we subtract 1 from the item number (*theItem–1*) in the routine shown in Example 5. If the active window is a system window (that is, a DA window), then *SystemEdit* will return false and the application will process the command as usual. Otherwise, *SystemEdit* will shuttle the event on to the DA for command processing and return true.

Resources for DAs

The code for the DA is not a *CODE* resource, as it is for applications, but is a *DRVR* because a DA is actually a device driver. The Macintosh resource compiler RMaker can be used to create a *DRVR* resource by reading the *CODE* resource created

1. AccEvent	An event (update, activate, keyboard, . . .)
2. AccRun	Do a periodic action
3. AccCursor	Change cursor shape
4. AccMenu	Menu selection
5. AccUndo	The Undo editing command
6. AccCut	The Cut editing command
7. AccCopy	The Copy editing command
8. AccPaste	The Paste editing command
9. AccClear	The Clear editing command

Table 1: *DA actions allowed by Ctl*

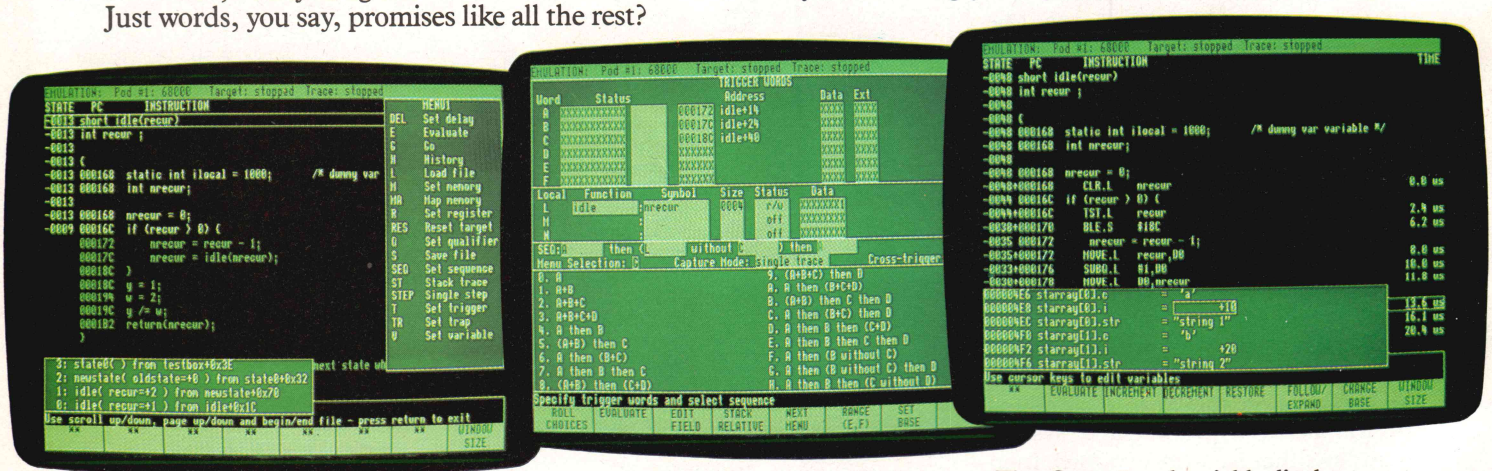


Promises, Promises.

Everybody promises, but nobody delivers a real-time, emulator-based C-debug environment like Arium's ECHO. 16-bit, true multitasking and UNIX®-based, ECHO gives you more power, speed and menu-driven features to handle the 68000 and other μPs better than the HP 64000, or anything else.

Just words, you say, promises like all the rest? Prove it to yourself. Read the screens below. Then ask any other development system—standalone or host control—to match them. We'll wait.

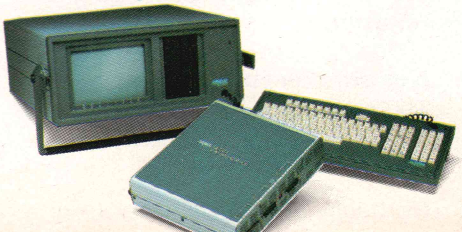
Now you know a few reasons (and there are plenty more) why ECHO should be your emulation tool for today's increasingly complicated software debugging.



Code Preview™ lets you see where your code is going. You can follow calls and branches (to 99 levels) on the screen, to select the source line on which to trigger, then set and break in one keystroke! The highlighted trace display (in source) and stack trace window show the path your program took.

Stack-Relative Trigger lets you trigger on the addresses and values of stack-relative variables—a "must" for effective C-debug where the address of an automatic variable is different each time the function is called and is determined at execution. Here, a read of the local variable "nrecur" is included in the trigger sequence.

TimeStamp™ **and variable display** are two further features that are a must for real-time C-debug. Note the display of two instances of a structure in array "starray." The contents of these structures, as for any C variable, can be changed right on the screen.



For a demonstration call
800/862-7486 (CA 714/978-9531)

ARIUM CORPORATION

1931 Wright Circle, Anaheim, CA 92806

UNIX is a registered trademark of AT&T.

CIRCLE NO. 114 ON READER SERVICE CARD

MAC DATA BASE
(continued from page 31)

by the linker with ID=1 and converting it to a *DRVR* resource. This is done by including the following lines in the resource file prior to compiling it with RMaker:

```
Type DRVR = PROC
  Deskacc,16
DeskaccFile
```

The name *DeskaccFile* is the linker's output file for the DA file, and *Deskacc* will be used as the DA name to appear under the Apple menu once the DA is installed in the system file. The resource ID then becomes the DA's driver reference number and is used by the operating system to implement the DA and is also used for owned resources.

Owned Resources

A range of 32 resource IDs has been reserved for each DA *DRVR* resource ID, so they are called owned resources. A special numbering convention is used to associate owned system resources with the resources they belong to. For any particular DA, this range is computed by adding $C000 and 32 times the driver reference number—for example, if the driver reference number is 16, then the range is –15,872 through –15,857.

When the DA is moved from its own file or a system file into a system file, all its resources for windows, menus, and so on must be transferred along with the code for the DA into the destination system file. If the destination system file already has a DA with the same *DRVR* resource ID, the Font/DA Mover will renumber it and all of its owned resources. Part of the DAMacMan resource file is shown in Listing One, beginning on page 48.

In summary, a DA program consists of at least three special procedures called *Open*, *Close*, and *Ctl*. The DA program may have other procedures as well, but it has no main body. You might think of the DA program as a module consisting of its own constants,g types, variables (Listing Two, page 48), and procedures (Listing Three, page 50). *Open* is called by *OpenDeskAcc* (from the running application); *Close* is called by *Ctl* (when the DA terminates itself) or *ExitToShell* (when the application terminates); and *Ctl* is called each time the application calls *SystemEvent*.

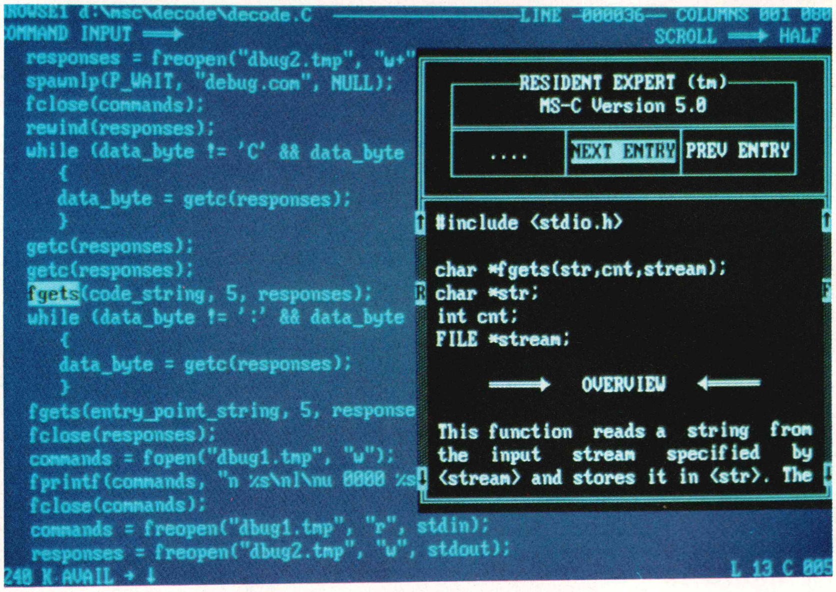
Listing Three shows these three procedures for DAMacMan, but keep in mind that these procedures must always exist for every DA even if they are tailored for some other purpose. In addition, because DAs are limited to 32K in size, the sophistication of a DA is restricted to miniature utility functions such as displaying the keyboard and so on. Implementing the MacMan database retrieval code was quite a challenge because of this limitation.

The Structure of DAMacMan

When DAMacMan is opened from the Apple menu, it looks in the system disk to see if the files it requires are present. If a file named Manual and another file named MMIndex are not present, DAMacMan will display an error dialog.

DAMacMan's related files are Manual, text from *Inside Macintosh* (the database); MMIndex, the index

RESIDENT EXPERT Pop-up Reference Guides...



Use One of Our's or Build Your Own!

THE POP-UP REFERENCE REVOLUTION BEGINS

How much development time could you save if you never had to open another PC language or technical reference manual again? What if you could just point at a compiler keyword, assembly instruction, or function name *on your screen* and with a keystroke have complete, authoritative information about language syntax, operands, parameters, examples, and much more?

INTRODUCING THE RESIDENT EXPERT SYSTEM

A growing library of comprehensive, disk resident reference guides about the PC *and* your favorite PC languages. All available instantly through our unique memory resident pop-up access system.

VIRTUALLY EVERYTHING YOU NEED TO KNOW

Each of our *Compiler Reference Guides* contains virtually everything you need to know to program with your *preferred implementation* of your favorite language. Language syntax, all library functions, compiler directives, and error codes are thoroughly documented.

Our *PC Programmer's Reference Guide* documents every PC (and AT) processor instruction and every BIOS and DOS service interrupt. You'll also find tables of keyboard codes, line drawing, ASCII, and IBM character sets, and much more.

THE SPECIALIST'S LIBRARY

Your compiler is unique. That's why our reference guides are *specialized*...each one designed for a particular vendor's language implementation.

NEW!!

RESIDENT EXPERT Compiler

Make your own Reference Guides

QUICK DRAW ACCESS SYSTEM

Point-and-shoot...just place the cursor over any term on your screen. Chances are we've got it fully detailed in one of our data bases.

Fully cross indexed...if the instruction or library function you're using isn't quite right, our related topics cross index can help you find a better one.

Multiple volumes on line...you can have one or a dozen of our pop-up reference guides on line...a complete library available instantly.

THE INFORMATION YOU NEED...WHERE YOU NEED IT

Our pop-up shell varies its size and shape dynamically, only taking as much space on your screen as it needs and it *never* covers your working area. You can see your work and our reference data at the *same* time.

RESIDENT EXPERT Shell (*) $19.95
(with PC-DOS/MS-DOS Reference Guide)

RESIDENT EXPERT Compiler..... $39.95
(create your own Reference Guides!)

RESIDENT EXPERT Reference Guides

Borland Turbo C (v1.0)	$19.95
Borland Turbo Pascal (v4.0)	19.95
Borland Turbo Prolog (v1.1)	19.95
Lattice C (v3.2)	39.95
Mark Williams Let's C (v4.0)	19.95
Microsoft C (v5.0)	39.95
Microsoft Quick C (v1.0)	19.95
PC Programmer's Reference Guide	$39.95

*The RESIDENT EXPERT Shell is required to access and display all Reference Guides....

SantaRita

For the location of your nearest Santa Rita Software dealer, or to order direct, call us at 1-214-727-9217. We'd like to hear from you.

Santa Rita Software
1000 E. 14th Street, Suite 365
Plano, Texas 75074

The RESIDENT EXPERT System

MAC DATA BASE
(continued from page 33)

into the database; MMSize.int, a temporary file used to generate a version of DAMacMan; MMConfig, a DAMacMan software tool for generating versions and MMLock, which locks and unlocks files.

Manual, the database, consists of two parts: the MacMan distribution text and the MacMan information that contains all the procedures and functions defined in *Inside Macintosh*, organized as follows:

```
\ name
   category#
   body
```

where *name* is the name of the procedure or function, *category* is the section of *Inside Macintosh* in which it is defined, and *body* is the information about the procedure or function.

MMLock is a MacMan tool whose job is to lock and unlock the files that are generated

The index file, MMIndex, contains records sorted with respect to name, each record having the following form:

```
Record
   name : String[25];
   start : longint;
   length : Integer;
   man : Integer;
end;
```

where *name* is the name of the procedure or function, *start* is the starting position relative to the beginning of the manual, *length* is the length of the text that belongs to this procedure or function, and *man* is an integer representing the section or manager of *Inside Macintosh* where this function or procedure is defined.

The MMSize.int file contains the statement:

```
Const MaxRec =    ;
```

The blank will be filled in by the MacMan tool MMConfig, prior to compiling MacMan. The value of *MaxRec* is equal to the number of entries in Manual. The DAMacman main program is shown in Example 6, page 41.

MacMan Tools

DAMacMan is configured and maintained through the use of a set of tools that must be applied each time the database is changed. MMConfig is a tool for constructing the index file for the database and the Pascal compiler *include* file that contains the size of the database. In addition, the integrity of the database is maintained by locking the database files using the MMLock program, described later.

MMConfig reads the file Manual and writes the ordered file MMIndex with records of the form:

```
Record
   name : String [25];
```

MAC DATA BASE
(continued from page 35)

```
  start : Longint;
  length : Integer;
  man : Integer;
end;
```

MMConfig performs the following steps:

1. It forms a record for each procedure or function in Manual.
2. It store a pointer to each of the records in an array of pointers.
3. It sorts the array with respect to name.
4. It creates the file MMIndex, to be read whenever MacMan is started up, and writes the sorted records to it.
5. It creates the file MMSize.int, which will be included during compilation of DAMacMan.
6. It locks the files Manual and MMSize.int so they cannot be opened by a user who is not supposed to have access to these files.
7. It stamps these files with the appropriate icons.

MMConfig is run to create MMIndex and MMSize.int after Manual has been edited the first time and whenever Manual is updated.

Finally, MMLock is a MacMan tool whose job is to lock and unlock the files that are generated by MMConfig.

Data Structures

As mentioned, after Manual has been constructed by entering all desired text into the database, MMConfig is run to generate the files MMIndex and MMSize.int. Then DAMacMan can be compiled and run. When DAMacMan is run, it loads the file MMIndex into a list of type *table*:

```
table : array [1. .MaxRec] of ptr1;
```

where *ptr1* is a pointer to a record similar to those stored in MMIndex and *MaxRec* is the number of functions or procedures in Manual. *MaxRec* is set by including the file MMSize.int.

Installing DAMacMan

DAMacMan is compiled into the resource file MacManFile. The Font/DA Mover utility tool is used to copy this file into a system file. After the DA has been installed in the system file of the start-up volume, it can be selected from the Apple menu.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, MACINTOSH, KAYPRO).

The complete source code and binary files for DAMacMan are also available on Macintosh disk (single-sided) from the authors.

Bibliography

Apple Computer Inc. *Inside Macintosh*. 4 vols. Reading, Mass.: Addison-Wesley, 1985–1986.

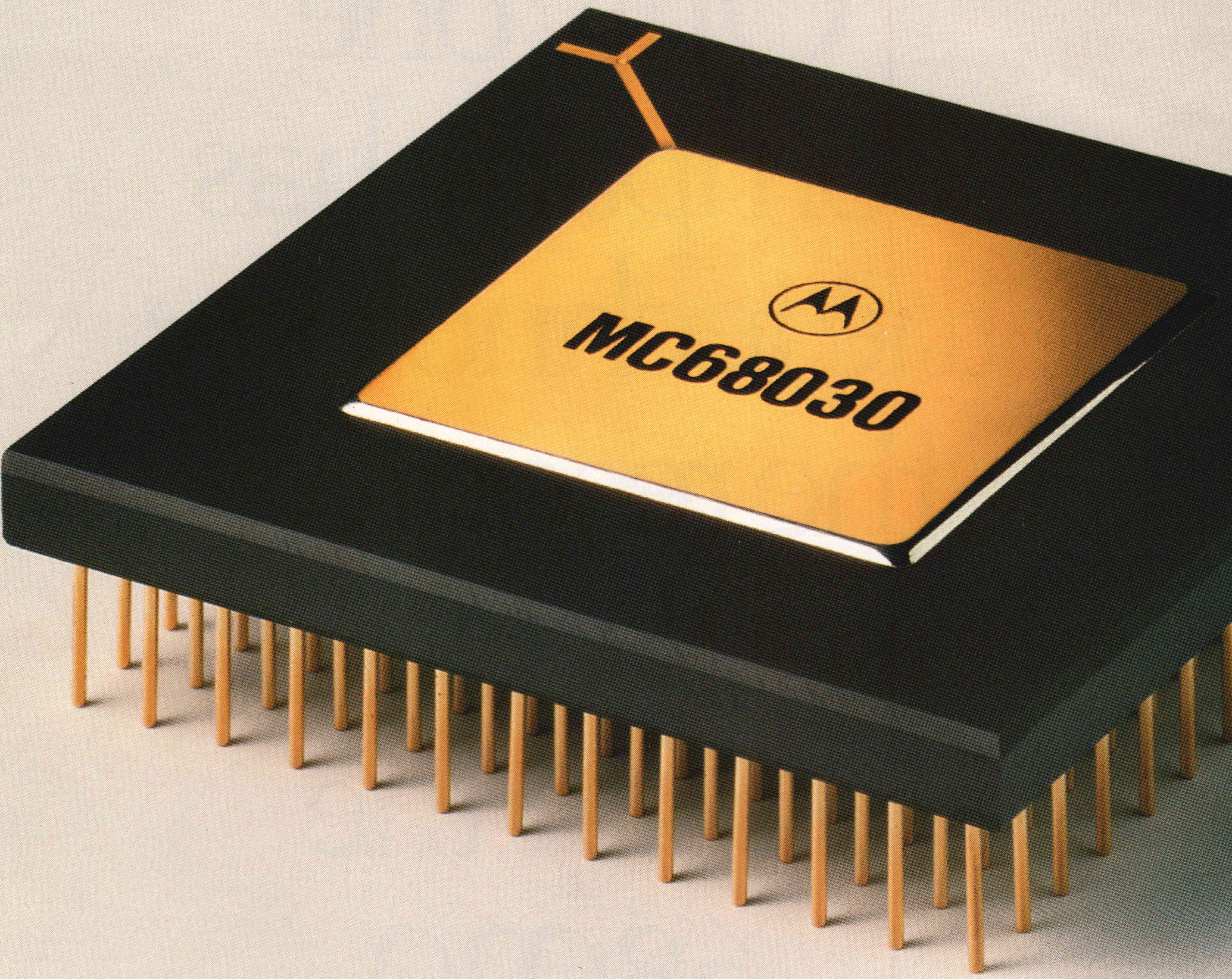
Chernicoff, Stephen. *Macintosh Revealed*, Volumes I and II. Hasbrouck Heights, N.J.: Hayden/Apple Press, 1985.

Reingold, E.; and Wilfred, H. *Data Structures*. Boston: Little, Brown, 1983.

Takatsuka, J.; Huxham, F; and Burnard, D. *Using the Macintosh Tool-*

Only one company has the technology, the resources, and the vision to top the 68020.

MOTOROLA



Introducing the 68030.

The next generation.

In 1984, we introduced the 68020. Now, three years later, it has the largest installed and broadest application base of any 32-bit MPU on the market. Having set that standard in the first place, we feel qualified to raise it. Which is why the only microprocessor that really surpasses the 68020 is our second generation 68030.

Meet the *Oh thirty!*

The '030 is twice the microprocessor its predecessor is. It's the first to sport an instruction cache, data cache and MMU on-chip. Combined with a Harvard-style parallel bus architecture that allows simultaneous, multiple fetches of instructions and data, processor throughput is pushed to unmatched levels.

What can you do with all that performance? Anything you like—from low-cost personal workstations to super-computers—and the 68030 will help you to do it less expensively.

With its burst fill mode for the dual caches, you'll be able to squeeze SRAM performance from low-cost DRAMs. It gives you graphics capability without the need for a graphics co-processor. And there's true object-code compatibility between the '030 and the '020. All this adds up to economies you can count on.

An architecture you can build on. And count on.

Application software that runs on any 68000 family MPU runs on the 68030. There's also a full array of development tools, and a new 68882 floating point co-processor, with up to 4x the performance of its predecessor. All of which gives your product plans an enormous amount of continuity. And that's not going to change. Since the 68030 supports both MS-DOS™ and UNIX® V.3, you can have your pick of over $12 billion worth of applications— and the broadest possible market.

With Motorola, you can see forever.

Our plans for the rest of the 68000 family extend well into the future, offering continuing compatibility and leading-edge performance. So you can go with Motorola, not just for what our microprocessors can do now, but what you'll be able to do with them later on.

For more information about the new 68030, call us toll-free at 800-521-6274 or write, Motorola Semiconductor Products, Inc., P.O. Box 20912, Phoenix, AZ 85036.

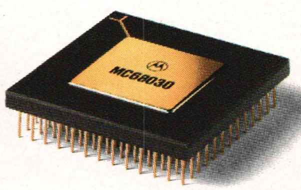
We're on your design-in team.

MOTOROLA

To: Motorola Semiconductor Products, Inc.
P.O. Box 20912, Phoenix, AZ 85036

345DRDJ010087

Please send more information on the *Oh thirty!*



Name ______

Title ______

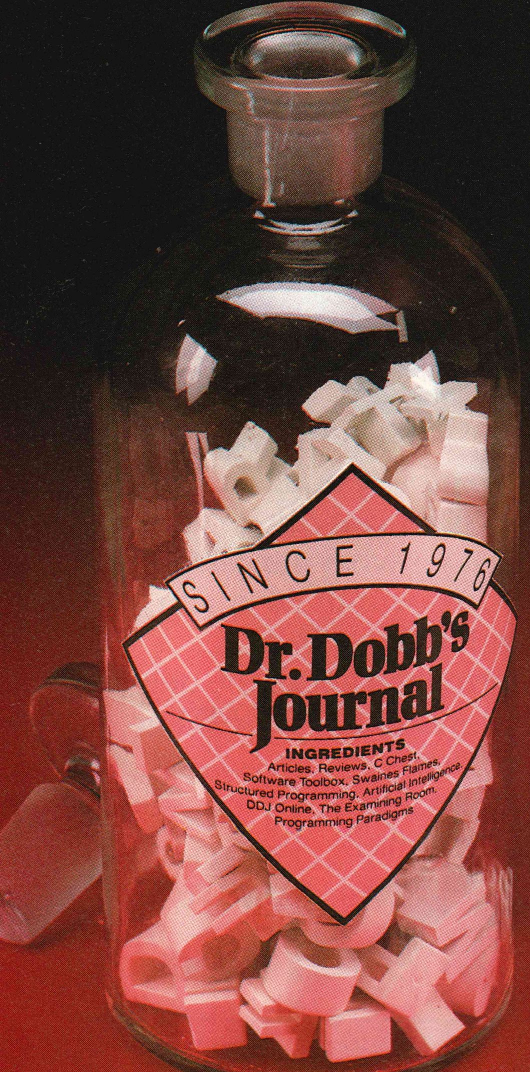
Company ______

Address ______

City ______ State ______ Zip ______

Call me (______) ______

THE CURE FOR COMMON CODE



Are you getting the recommended monthly allowance of C, Assembly, Forth, Pascal, BASIC or Modula-2? Subscribe to *Dr. Dobb's Journal of Software Tools* and you won't catch any nasty bugs again!

Each month the Doctor brings you aid for ailing algorithms and the cure for common code. For the latest developments in software design and pages of code that will make you a more productive programmer, take the Dr. Dobb's prescription.

For more than a decade, the programming elite have known *Dr. Dobb's Journal* to be the foremost source of software tools. Subscribe now and get your monthly dose from the Doctor.

Dr. Dobb's Journal of Software Tools

The Rx for Programmers

Subscribe Now & Save Over 15% Off the Newsstand Price!

$5 SAVINGS

SUBSCRIBE AND SAVE!
Subscribe to

DR. DOBB'S JOURNAL OF SOFTWARE TOOLS
and save over $5—a 15% savings off the cover price!

☐ **1 year $29.97** ☐ **2 years $56.97**

☐ Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______ Exp. date ______

Signature ______

Name ______

Address ______

City ______ State ______ Zip ______

ONLY $29.97! YOU SAVE OVER $5.00

Savings based on a full one-year cover price of $35.40. Canada & Mexico add $10 for surface mail per year. All other countries add $27 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. **3467**

$5 SAVINGS

SUBSCRIBE AND SAVE!
Subscribe to

DR. DOBB'S JOURNAL OF SOFTWARE TOOLS
and save over $5—a 15% savings off the cover price!

☐ **1 year $29.97** ☐ **2 years $56.97**

☐ Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment enclosed ☐ Bill me later

Card # ______ Exp. date ______

Signature ______

Name ______

Address ______

City ______ State ______ Zip ______

ONLY $29.97! YOU SAVE OVER $5.00

Savings based on a full one-year cover price of $35.40. Canada & Mexico add $10 for surface mail per year. All other countries add $27 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

A Publication of M & T Publishing, Inc. **3467**

COMMENTS & SUGGESTIONS

January 1988, #135

Dear Reader,
Dr. Dobb's has a long tradition of listening to its readers. We like to hear when something really helps or, for that matter, bothers you. In this hectic world of ours, however, it is often difficult to take time to write a letter. This card provides you with an easy way to correspond and, if you include your name and address, we may use appropriate comments in The Letters column. Simply fill it out and drop it in the mail.

—Ed

Which articles or departments did you enjoy the most this month? Why?

______

Comments or suggestions ______

Name: ______

Address: ______

Dr. Dobb's Journal of Software Tools

The Rx for Programmers

Subscribe Now & Save Over 15% Off the Newsstand Price!

MAC DATA BASE
(continued from age 36)

box with C. Berkeley, Calif.: Sybex, 1986.
TML Systems. *MacLanguage Series Pascal User's Guide and Reference Manual.* Jacksonville, Fla.: TML Systems, 1985.
Wirth, Niklaus. *Algorithms + Data Structures = Programs.* Englewood Cliffs, N.J.: Prentice-Hall, 1976.
Wootton, Alan. "Resource Utility DA with TML." *MacTutor* (November 1985).

DDJ

(Listings begin on page 48.)

Vote for your favorite feature/article.
Circle Reader Service **No. 7.**

```
PROGRAM  DAMacMan;
{ DAMacMan is an online inside macintosh, running as a desk accessory

  Pascal source:     DAMacMan.pas     Main Program
  Uses    :

            MMSize.Int       The Manual Size, Generated
                                              by MMConfig Tool
            DAMacMan.int   The Interface include file
            DAOthers.imp     Our own procedures and functions
            DAThree.imp       The open, Ctl, Close procedures
            DAIndex          Index To Database Entries
         Manual              Database from Inside Macintosh
  Resources  :       DAMacMan.R

  Creation Date:  July 1St, 1986
  Author      :          Abdullah  Al-Dhelaan
                       (TGL Software Development Group )
                      Oregon State University
}
{ The next four include files below are Interface to Toolbox ...}
{$I MemTypes.ipas    }
{$I QuickDraw.ipas    }
{$I OSIntf.ipas           }
{$I ToolIntf.ipas        }

{$I MMSize.Int          }     {The Manual Size, Generated
                                              by MMConfig Tool }
{$I DAMacMan.int      }    { The Interface include file    }
{$I DAOthers.imp        }    { Our own procedures and functions }
{$I DAThree.imp         }    {The open, Ctl, Close procedures }

{--------------------------Main Program---------------------------}
BEGIN
        { Desk Accessory,  There should be no main program }
END.
```

Example 6: *The DAMacMan main program*

Listing Twelve *(Text in December.)*

EXTERNS.H

```
/*
   This include file contains all of the external symbol definitions
   for modules that refer to external data.
*/

#ifndef  GLOBALS

extern   unsigned int   load_segment ;
extern   SEG_DESCRIPTOR *seg_list ;

extern   char   config_fname[FILENAME], abs_fname[FILENAME] ;
extern   char   module_name[FILENAME], locate_fname[FILENAME] ;
extern   char   print_fname[FILENAME], exe_fname[FILENAME] ;
extern   char   map_fname[FILENAME], tmp_fname[FILENAME] ;

extern   char   command_line[COMMAND_LINE] ;

extern   int    tmp_file ;
extern   int    abs_file ;
extern   int    exe_file ;

extern   FILE   *map_file ;
extern   FILE   *print_file ;
extern   FILE   *config_file ;

extern   BOOLEAN  boot_rec, help, config, user_abort, hex_name ;


#endif
```

End Listing Twelve

Listing Thirteen

GLOBALS.H

```
/*
   This module contains all of the global data declarations
*/

#ifndef  GLOBALS

unsigned int   load_segment ;
SEG_DESCRIPTOR *seg_list ;

char   config_fname[FILENAME], abs_fname[FILENAME] ;
char   module_name[FILENAME], locate_fname[FILENAME] ;
char   print_fname[FILENAME], exe_fname[FILENAME] ;
char   map_fname[FILENAME], tmp_fname[FILENAME] ;

char   command_line[COMMAND_LINE] ;

int    tmp_file ;
int    abs_file ;
int    exe_file ;

FILE   *map_file ;
FILE   *print_file ;
FILE   *config_file ;

BOOLEAN  boot_rec = FALSE, help = FALSE, config = FALSE ;
BOOLEAN  hex_name = FALSE, user_abort = FALSE ;

#define  GLOBALS

#endif
```

End Listing Thirteen

Listing Fourteen

LOC.H

```
#define  MAX_TOKEN      80
#define  MAX_LINE       80
#define  FILENAME       80
#define  COMMAND_LINE   256

#define  PAGE_SIZE      512

#define  ERROR     0      /* Error condition exists */
#define  OK        1      /* No error(s) detected */

#define  T_EOL    -2      /* Reached the end of a line */
#define  T_ERROR  -1      /* Detected an error condition */
#define  T_OK      0      /* Everything fine */
#define  T_OCT     1      /* Octal format integer */
#define  T_DEC     2      /* Decimal format integer */
```

```
#define T_HEX     3       /* Hexidecimal format integer */
#define T_WORD    4       /* Symbol character string */
#define T_OP      5       /* Operator character string */

typedef enum  { FALSE = 0, TRUE }  BOOLEAN ;

#define low_byte(x)       ((x) & 0xff)
#define high_byte(x)      (low_byte((x) >> 8))
#define dim(x)            (sizeof(x)/sizeof(x[0]))

typedef struct {
      unsigned int   signature ;
      unsigned int   length ;
      unsigned int   pages ;
      unsigned int   reloc_items ;
      unsigned int   header_size ;
      unsigned int   min_para ;
      unsigned int   max_para ;
      unsigned int   stack_seg_disp ;
      unsigned int   initial_sp ;
      unsigned int   checksum ;
      unsigned int   initial_pc ;
      unsigned int   code_seg_disp ;
      unsigned int   first_reloc_item ;
      unsigned int   overlay_num ;
   } EXE_HEADER ;


typedef struct sym_list  {
            struct sym_list *next ;
            char  name[32] ;
            unsigned int   value ;
            unsigned char  type ;
         } SYMBOL_LIST ;

typedef struct  seg_descriptor  {
            struct seg_descriptor    *next ;
            long  position ;
            char  name[32] ;
            char  class[32] ;
            unsigned int   vseg ;
            unsigned int   pseg ;
            unsigned int   offset ;
            unsigned int   len ;
            int   inited ;
            int   romable ;
            unsigned int   symbols ;
            SYMBOL_LIST *symbol_list ;
         } SEG_DESCRIPTOR ;

char  *load_exe_file() ;
SEG_DESCRIPTOR *build_seg_list() ;

int   get_ch() ;
int   unget_ch() ;
char  *get_cmd() ;
char  *get_line() ;

int get_token() ;

char  *get_mem(unsigned long) ;
void  free_mem(char *) ;

char  *process_command_line() ;
void  read_symbol_table(SEG_DESCRIPTOR *) ;

void  output_hex_OMF(int, SEG_DESCRIPTOR *, unsigned char *) ;
void  open_file_system() ;
void  close_file_system() ;
void  break_handler() ;

#define  DATA_RECORD     0
#define  EOF_RECORD      1
#define  ADDR_RECORD     2
#define  START_RECORD    3

void  write_START_record(int, unsigned char *) ;
void  write_ADDR_record(int, unsigned int ) ;
void  write_DATA_record(int, unsigned int, unsigned char *,
                                  unsigned int ) ;
void  write_EOF_record(int) ;
void  output_hex_record(int, unsigned char, unsigned int,
                                  unsigned char *, unsigned char) ;;
```

End Listing Fourteen

Listing Fifteen

```
          page    60, 132
          name    startup
          title   LOCATE Example ROM System Startup Code
          subttl  Turbo C 1.0 Version
```

(continued on next page)

Listing Fifteen *(Listing continued)*

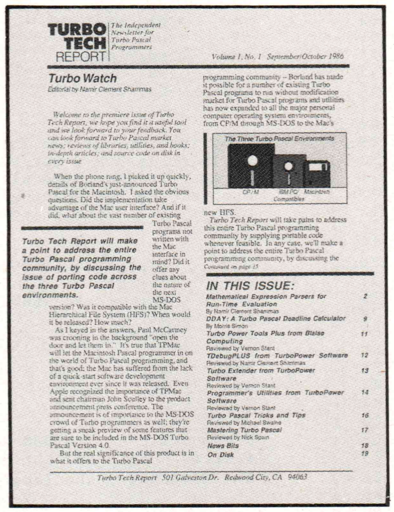
```
;
;       ROM System Startup Code for Turbo C 1.0
;       Copyright (C) 1987 by Rick Naro.  All rights reserved.
;

;
;       Segment and group declarations
;

_text   segment byte    public  'CODE'
_text   ends
_etext  segment para    public  'CODEEND'
_etext  ends
_data   segment para    public  'DATA'
_data   ends
_bss    segment para    public  'BSS'
_bss    ends
_bssend segment byte    public  'BSSEND'
_bssend ends
_stack  segment para    stack   'STACK'
_stack  ends

dgroup  group   _data, _bss, _bssend

assume  cs:_text, ds:dgroup, ss:_stack

;
;       This version of the startup code expects 32-bit code pointers.
;       (medium or large memory models)
;

extrn   _main : far     ; Change to near for small/compact memory model

_text   segment

public  start

start   proc    far

        cli                             ; Disable interrupts

;
```

(continued on page 46)

Turbo Tech Report Speaks Your Language.

Turbo Pascal Articles and Reviews

News and commentary



TURBO TECH REPORT | The Independent Newsletter for Turbo Pascal Programmers

Volume 1, No. 1 September/October 1986

Turbo Watch

Editorial by Namir Clement Shammas

Welcome to the premiere issue of Turbo Tech Report, we hope you find it a useful tool and we look forward to your feedback. You can look forward to Turbo Pascal market news; reviews of libraries, utilities, and books; in-depth articles; and source code on disk in every issue

When the phone rang, I picked it up quickly, details of Borland's just-announced Turbo Pascal for the Macintosh. I asked the obvious questions. Did the implementation take advantage of the Mac user interface? And if it did, what about the vast number of existing Turbo Pascal programs not written with the Mac interface in mind? Did it offer any clues about the nature of the next MS-DOS version? Was is compatible with the Mac Hierarchical File System (HFS)? When would it be released? How much?

As I keyed in the answers, Paul McCartney was crooning in the background "open the door and let them in." It's true that TPMac will let the Macintosh Pascal programmer in on the world of Turbo Pascal programming, and that's good; the Mac has suffered from the lack of a quick-start software development environment ever since it was released. Even Apple recognized the importance of TPMac and sent chairman John Sculley to the product announcement press conference. The announcement is of importance to the MS-DOS crowd of Turbo programmers as well; they're getting a sneak preview of some features that are sure to be included in the MS-DOS Turbo Pascal Version 4.0.

But the real significance of this product is in what it offers to the Turbo Pascal programming community – Borland has made it possible for a number of existing Turbo Pascal programs to run without modification market for Turbo Pascal programs and utilities has now expanded to all the major personal computer operating system environments, from CP/M through MS-DOS to the Mac's new HFS.

Turbo Tech Report will make a point to address the entire Turbo Pascal programming community, by discussing the issue of porting code across the three Turbo Pascal environments.

The Three Turbo Pascal Environments

CP/M | IBM PC/Compatibles | Macintosh

Turbo Tech Report will take pains to address this entire Turbo Pascal programming community by supplying portable code whenever feasible. In any case, we'll make a point to address the entire Turbo Pascal programming community, by discussing the

Continued on page 15

IN THIS ISSUE:

Mathematical Expression Parsers for Run-Time Evaluation By Namir Clement Shammas	2
DDAY: A Turbo Pascal Deadline Calculator By Morris Simon	9
Turbo Power Tools Plus from Blaise Computing Reviewed by Vernon Stant	11
TDebugPLUS from TurboPower Software Reviewed by Namir Clement Shammas	12
Turbo Extender from TurboPower Software Reviewed by Vernon Stant	13
Programmer's Utilities from TurboPower Software Reviewed by Vernon Stant	14
Turbo Pascal Tricks and Tips Reviewed by Michael Swaine	16
Mastering Turbo Pascal Reviewed by Nick Spain	17
News Bits	18
On Disk	19

Turbo Tech Report 501 Galveston Dr. Redwood City, CA 94063

A disk filled with Turbo Pascal code!

The newsletter/disk publication for Turbo Pascal® users

Are you looking for powerful utilities written in Turbo Pascal that you can use to develop software or incorporate into your programs? Are you interested in improving and expanding your Turbo Pascal programming skills?

Then you deserve a subscription to *Turbo Tech Report*, the bimonthly newsletter/disk publication from the publishers of *Dr. Dobb's Journal* and *Micro/Systems Journal*. Each issue delivers more than 250K of Turbo Pascal source code programs on disk, and 24+ pages of articles, Turbo Pascal software and book reviews, and analysis and commentary. It's the only publication delivering such focused technical articles with code on disk. Each valuable issue contains:

- ***Articles*** on topics like speedy 3D graphics, mathematical expression parsers, creating global gotos, memory resident and AI applications and more—all written by Turbo experts.
- ***Reviews*** of the latest Turbo Pascal software programs from companies like Borland International, Blaise Computing, Media Cybernetics, Nostradamus, and more!
- ***News and commentary*** detailing the latest products and developments in the Turbo Pascal programming community.
- ***A disk filled with Turbo Pascal code!*** You'll get the Turbo Pascal utilities and routines discussed in the newsletter's articles, as well as applications developed by Turbo users from around the world. You'll receive programs that make labels, generate menus, provide faster screen access, transfer files, and more!

If you're an expert Turbo Pascal programmer or a novice interested in expanding your Turbo skills, you need a publication that speaks your language: *Turbo Tech Report*. Subscribe today at the special price of just $99—that's 33% off the regular price of $150. To order by credit card, call toll-free 1-800-533-4372. Or mail the coupon with your payment to *Turbo Tech Report*, 501 Galveston Drive, Redwood City, CA 94063.

—**Yes!** I want a one-year subscription to *Turbo Tech Report* (6 issues with 6 disks) for $99.

Format: ☐ PC/MS-DOS ☐ Macintosh

CP/M: ☐ Kaypro ☐ Osborne ☐ Apple

PAYMENT MUST ACCOMPANY ALL ORDERS

☐ Check/money order enclosed.

☐ Charge my: ___ VISA ___ M/C ___ AmExp.

Card # __________ Exp. ______

Signature __________

Name __________

Address __________

City ______ State ____ Zip ____

Orders outside U.S.: add $30.

CIRCLE NO. 121 ON READER SERVICE CARD

Listing Fifteen *(Listing continued.)*

```
;       Initialize the stack segment and pointer
;

        mov     ax, _stack              ; Get the stack segment value
        mov     ss, ax                  ; Put in SS
        mov     sp, offset tos          ; Load the TOS in SP

;
;       Set up the segment registers for the initialization of DGROUP.
;       ES is initialized to DGROUP and DS is initialized to the copy of
;       DGROUP in ROM.
;

        mov     ax, dgroup              ; Get the segment for DGROUP
        mov     es, ax                  ; Install in ES

        mov     ax, _etext              ; Get the _etext segment
        inc     ax                      ; Adjust for the size of _etext
        mov     ds, ax                  ; Install in DS

;
;       Copy the copy of the initialized data segment _DATA from its
;       position in ROM (just after the CODE class) to the real _DATA
;       segment in RAM.  The size of the _DATA segment is computed by
;       subtracting the start of _DATA (the label idata) from the start
;       of the next segment (the label bdata).
;

        mov     si, offset DGROUP:idata ; Starting offset of _DATA
        mov     di, si                  ; Same offset, different segment
        mov     cx, offset DGROUP:bdata ; Get the end offset
        sub     cx, di                  ; Subtract the two for a length
        rep     movsb                   ; Copy initialized data

        push    es                      ; Get the value of DGROUP
        pop     ds                      ; Now it is in DS

;
;       C expects that the BSS segment is cleared when the program is
;       started.  Zero out the BSS segment within DGROUP by computing
;       the size using the label bdata and edata labels
;

        xor     al, al                  ; Use a zero fill pattern
        mov     di, offset DGROUP:bdata ; Get the starting offset
        mov     cx, offset DGROUP:edata ; Get the ending offset
        sub     cx, di                  ; Subtract the two for a length
        rep     stosb                   ; Zero out the BSS

;
;       Call main and hopefully never return ...
;

        sti                             ; Re-enable interrupts
        call    _main                   ; Call main()

;
;       Handle a return by restarting the entire process
;

        jmp     start                   ; Start all over

start   endp

_text   ends

        page
_etext  segment
public  tend
        db      16 dup (?)              ; Force alignment of the label
                                        ; tend with the next paragraph
tend    label   byte                    ; Mark the end of the segment
                                        ; Do NOT change this segment
_etext  ends

_data   segment
public  idata
idata   label   byte                    ; Start of the _DATA segment
_data   ends

_bss    segment
public  bdata
bdata   label   byte                    ; Start of the _BSS (and the
                                        ; end of the _DATA segment
_bss    ends

_bssend segment
public  edata
edata   label   byte                    ; Mark the end of the BSS
_bssend ends

_stack  segment
public  tos                             ; Make the TOS public
        dw      256 dup (?)             ; Declare the stack size
tos     label   byte                    ; Define the top of stack
_stack  ends

        end     start
```

End Listings

Listing One *(Text begins on page 24.)*

```
*           DAMacMan.r                    Resource
*           July 3, 1986                  Last modified
*
* Resource file for "DAMacMan.pas" for use with MacLanguage
* series Pascal Compiler (TML Pascal)
*
*
* all resources must be between -15872 and -15841 inclusive
* if they are to travel  with DRVR number 16

MacManFile
DFILDMOV

TYPE DRVR = PROC
  MacMan,16
DAMacMan

* this DLOG is the main window.
* the StatText items are used only for their rectangles.

Type WIND
  ,-15872
Untitled
50 40 300 510
InVisible GoAway
0
0

* menu used by the DA

type MENU
  ,-15872
Macman        ;; menu title
about Macman
(-
Find By Name
View By Category

* This second dialog is for about menu item
* the button and icon do nothing. Just decoration

Type ALRT
  ,-15872
50 50 330 430
-15872
4444

  ,-15871
50 50 300 480
-15871
4444

  ,-15870
60 80 126 430
-15870
4444
```

End Listing One

Listing Two

```
{
            DAMacMan.int                    Interface
            July 3, 1986                        Last modified

   An interface file that Contains all the needed types and vars

}

Const

  accEvent   = 64;
  accRun       = 65;
  accCursor = 66;
  accMenu    = 67;
  accUndo        = 68;
  accCut       = 70;
  accCopy     = 71;
  accPaste    = 72;
  accClear    = 73;

  ManFile      = 'Manual';     {Name of Database File}
  Vnum         = 0;  {default drive}

 Type
    Str25   = String [25];
    Lptr     = ^LongInt;
    Item    =  Record                            { An entity record}
                name        : Str25;                { Proc/Func. Name }
                   start             : LongInt;          { Starting pos. on the Manual }
               length        : Integer;            { Length of Proc./Func. Text }
               man           : Integer;            { Category # }
           END;
```

```
      ptrl   =   ^ Item;
      myarray   = ARRAY [1..MaxRec] of Ptrl;  { The main array }

  {This is the data. A handle to it will be stored in the dctl storage field
    of the DCltEntry Record  }

    GlobalsH = ^GlobalsP;
    GlobalsP = ^GlobalsRec;
    GlobalsRec = Record { DAMacMan Database Info }
                           hScroll    : ControlHandle;
                                                     {Horizontal scroll for window.}
                           vScroll    : ControlHandle;{Vertical scroll for window}
                           pRect      : Rect;        {Rectangle within window to see}
                        tRect      : Rect;
                        hTE         : TEHandle;      {Text is here...}
                           table      : myarray;     {Index to entries}
                    noIndex    : boolean;            {Error if no Index on Disk}
                       noman      : boolean;         {Error if no Database on Disk}
                      END;

  { This is used to store system info about the state of the driver. It will
    be passed to us on all calls from system. This Record is definded in the
    interface (TML files) above (as DCtlEntry) . it is not strictly necessary to
    redefine it here. But doing so,  will enable me to use dctlStorage^^ to
    refer to GlobalsRec without coercing Types }

    MyDeviceEntry = Record
                DctlDriver   : HAndle;     { Pointer to driver }
                DctlFlags     : Integer;  { Flags }
                DctlQueue   : Integer;   { Low-order byte, drivers version number }
                DctlQhead   : Lptr;          { Pointer to first entry in drivers I/O queue }
                DctlQTail      : Lptr;         { pointer to last entry in drivers I/O queue }
                DctlPosition: LongInt;  { Byte position }
                DctlStorage : GlobalsH; { Handle to RAM driver's private storage }
                DctlRefNum : Integer;  { Driver's reference number }
                DctlCurTicks: LongInt;  { Used internally by Device Manager }
                DctlWindow : Grafptr;  { Pointer to driver's window Record }
                DctlDelay      : Integer;  { number of ticks between periodic actions }
                DctlEmask   : Integer;   { Desk accessory event mask }
            DctlMenu      : Integer;   { Menu ID of menu associated with driver }
             END;

  { No VARiables for main program are allowed }
```

End Listing Two

(Listing Three begins on next page)

Listing Three *(Listing continued, text begins on page 24.)*

```
{
          DAThree.imp                  Implementatioon
          July 3, 1986                  Last modified

      THE FOLLOWING PROCEDURES SHOULD BE IN EVERY DESK ACCESSORY
      AND THEY ARE CALLED BY THE SYSTEM

}


 PROCEDURE open    ( VAR Device : MyDeviceEntry;
                       VAR block   :  ParamblockRec);
 { Open Makes a window and sets-up our private storage.
   we may get an open call even after we are already open }

CONST
        InfoLength = 700;   {The number of chars. fro the distribution text}
   VAR
     Wpk        : WindowPeek;
     Dtyp       : Integer;
     ID           : Integer;
     Dhan       : Handle;
     tmpPtr   : ptr;
     dummy    : GlobalsRec;
     Watch      : CursHandle;   {Handle to wristwatch cursor}

   BEGIN

   {Use ID for all Resource access}
   ID := $C000 - 32 * (1 + Device.dctlRefNum);
   Device.dctlMenu := ID;
    With Device do
    if DctlWindow = nil
   THEN BEGIN
      { Create a hole in the heap.  It is good practice to keep
           Window records off of the bottom of the Application Heap. }
        DctlStorage := pointer(NewHandle(SizeOf(dummy)));
        TmpPtr := NewPtr($1000);
        Hlock(Handle(DctlStorage));
        With DctlStorage^^ do
         BEGIN {initialize our storage }
```

```
          noindex := false;  {Index found}
             noman    := false;   {Manual found}
             Watch := GetCursor (WatchCursor);
             SetCursor (Watch^^);                     {Indicate delay }
           CreateWindow (Device,ID);
            { post information }
          DoOpen (Device,'MacMan Distribution',0,InfoLength);
              If not noman
              Then LoadData(device); {Load the array}
          initcursor;
      END;  { of with storage }
         {Deallocate our temporary pointer }
      Hunlock(Handle(DctlStorage));
       DisposPtr(TmpPtr);
    END; { of if }
 END; { of open }

{----------------------------------------------------------------------}

PROCEDURE close   (   VAR Device : MyDeviceEntry;
                         VAR block    : paramBlockRec);
 BEGIN
    deactivate(Device);       { remove menu }
    with Device do
    BEGIN
      disposHandle(Handle(DctlStorage));        { kill data }
      disposeWindow(DctlWindow);                 { erase window }
       DctlWindow := nil;
   END; {of with }
 END; { of close }

{----------------------------------------------------------------------}

 PROCEDURE ctl (      VAR Device : MyDeviceEntry;
                      VAR block : ParamBlockRec) ;
{ Here is the main entry point for system calls. The permanent bolck tell us
  what the nature of the call is. }
  VAR
       mousept      : point;
       wpnt         : Grafptr;
       item         : Integer;
       ibeam        : cursHandle;
       ignore       : Integer;
       longignore   : LongInt;
```

(continued on next page)

Listing Three *(Listing continued, text begins on page 24.)*

```
      BEGIN
         setport(Device.DctlWindow);
         Hlock(Handle(Device.DctlStorage));
         with Device, DctlStorage^^,block do
          BEGIN
             TEidle(hTE);
             CASE csCode of
                 accevent    :  Event(Device, block);
                 accCursor :
                   BEGIN
                      ibeam := GetCursor(ibeamcursor);
                      GetMouse(mousePt);
                      If (PtInRect(mousePt,pRect))
                        THEN SetCursor(iBeam^^)
                        ELSE InitCursor;
                   END;
                accmenu    :     { CASE out menu item number }
                  BEGIN
                     Initcursor;
                     CASE csParam[1] of
                             1 : Doabout (Device.dctlmenu);    {about... }
                            3 : IF (not noindex) THEN  DoFind  (Device,'');
                            4 : IF (not noindex) THEN  DoView  (Device)   ;
                        END; { of CASE menu }
                     HiliteMenu(0);
                 END; { of menu CASE }
              accCopy :
                BEGIN
                         longignore := ZeroScrap;  {Init. Scrap}
                  TECopy(hTE);            {Copy text from hte to TextEdit scrap}
                  ignore := TEToScrap;  {Copy  TextEdit scrap to desk scrap }
                  ignore := UnloadScrap; {Copy desk scrap to file scrap}
               END;
            Otherwise ;
          END; { CASE of ... }
         END; { of with block }
       Hunlock(Handle(Device.DctlStorage));
      END; { of control  PROCEDURE }
```

End Listings

"How to protect your software by letting people copy it."

By Dick Erett, President of Software Security

Inventor and entrepreneur, Dick Erett, explains his company's view on the protection of intellectual property.

"A crucial point that even sophisticated software development companies and the trade press seem to be missing or ignoring is this:

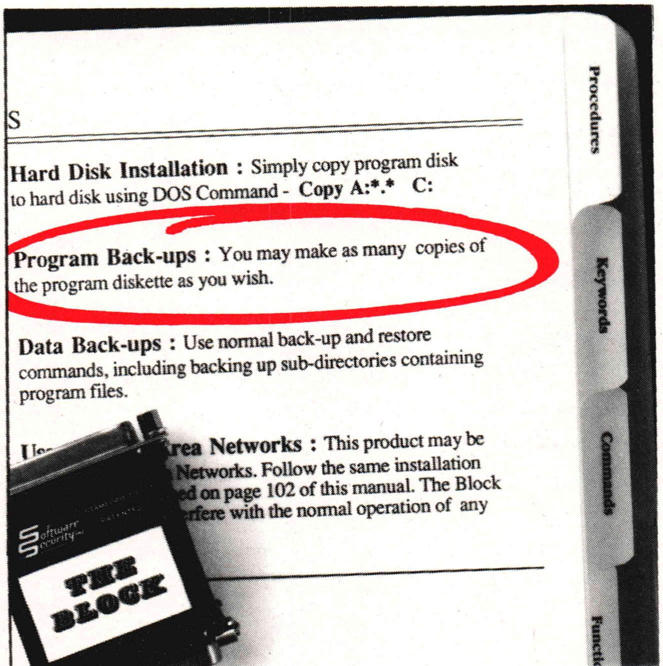
Software protection must be understood to be a distinctively different concept from that commonly referred to as copy protection.

Fundamentally, software protection involves devising a method that prevents unauthorized use of a program, without restricting a legitimate user from making any number of additional copies or preventing program operation via hard disk or LANs.

Logic dictates that magnetic media can no more protect itself from misuse than a padlock can lock itself.

Software protection must reside outside the actual storage media. The technique can then be made as tamper proof as deemed necessary. If one is clever enough, patent law can be brought to bear on the method.

Software protection is at a crossroads and the choices are clear. You can give product away to a segment of the market, or take a stand against the theft of your intellectual property.



Soon all software installation procedures will be as straightforward as this. The only difference will be whether you include the option to steal your product or not.

"...giving your software away is fine..."

We strongly believe that giving your software away is fine, if you make the decision to do so. However, if the public's sense of ethics is determining company policy, then you are no longer in control.

We have patented a device that protects your software while allowing unlimited archival copies and uninhibited use of hard disks and LANs. The name of this product is The BLOCK™.

The BLOCK is the only patented method we know of to protect your investment. It answers all the complaints of reasonable people concerning software protection.

In reality, the only people who could object are those who would like the option of stealing your company's product.

"...eliminating the rationale for copy-busting..."

Since The BLOCK allows a user to make unlimited archival copies the rationale for copy-busting programs is eliminated.

The BLOCK is fully protected by federal patent law rather than the less effective copyright statutes. The law clearly prohibits the production of work-alike devices to replace The BLOCK.

The BLOCK attaches to any communications port of virtually any microcomputer. It comes with a unique customer product number programmed into the circuit.

The BLOCK is transparent to any device attached to the port. Once it is in place users are essentially unaware of its presence. The BLOCK may be daisy-chained to provide security for more than one software package.

Each software developer devises their own procedure for accessing The BLOCK to confirm a legitimate user. If it is not present, then the program can take appropriate action.

"...possibilities... limited only by your imagination..."

The elegance of The BLOCK lies in its simplicity. Once you understand the principle of The BLOCK, hundreds of possibilities will manifest themselves, limited only by your imagination.

Your efforts, investments and intellectual property belong to you, and you have an obligation to protect them. Let us help you safeguard what's rightfully yours. Call today for our brochure, or a demo unit."

Software Security inc.

870 High Ridge Road Stamford, Connecticut 06905

203 329 8870

Listing One *(Text begins on page 90.)*

```
/*------------------------------ file information ----------------------------*/



/*
       custom controls demo.c

              c source code file for a minimal Mac program that demonstrates
                     controls drawn with a custom CDEF resource

              the custom CDEF resource that's demonstrated provides 16 button
variations
              the buttons …
                            ° … live in a rectangular space
                            ° … can be outlined, shadowed, or bare
                            ° … can contain text in any font-style-size, an icon, or
a picture
                            ° … can indicate highlighting via inversion or a change
of content

       edited and compiled with Lightspeed C 2.13

       written and ©1987 by Stan Krute. all rights reserved. no part of this file,
       or the object code it leads to, may be reproduced, in any form or by any means,
       without        the express written permission of the author and copyright
holder.

       timestamp:            3:49 pm PST                         November 16, 1987
       spacestamp:           18617 Camp Creek Road       Hornbrook, California
96044

       this file looks good in 9 point Courier, LSC tabs set to 3
*/

/*--------------------------------- include files -----------------------------*/

/* definitions for Mac OS managers used herein */
#include       "ControlMgr.h"
#include       "DialogMgr.h"
#include       "EventMgr.h"
#include       "FontMgr.h"
#include       "MenuMgr.h"
#include       "Quickdraw.h"
#include       "StdFilePkg.h"

/* our stuff */
#include       "custom controls demo.h"           /*   private definitions for this
file  */


/*----------------------------- main program block ----------------------------*/

void    main()

              {
              /* local variable */
              int            theItem ;

              /* initialize Mac OS managers  */
              initializeManagers() ;

              /* see what the world is like */
              studyAndSetEnvironment () ;

              /* set up and draw a (dummy) title menu */
              InsertMenu( GetMenu(titleMenuID), append ) ;
              DrawMenuBar() ;

              /* set up and draw a modal dialog window */
              getThatDialogCookin () ;

              /* initalize our doneness indicator */
              finished = false ;

              /*  run the main event loop    */
              do

                     {
                     ModalDialog (noFilterProcedure, &theItem) ;
                     dealWithDialogItem (theItem) ;
                     }
              while

                     ( ! finished ) ;

              /*  leave neatly when done */
              DisposDialog (ourDialog) ;                /* bye bye to dialog */
              ExitToShell() ;                                  /* bye bye to
program */
              }

/*--------------------------- initializeManagers -----------------------------*/

/*   initialize the heap, cursor, and Mac Operating System managers     */
```

(continued on page 57)

Support your clients without leaving your office!

For $99 pcAnywhere gives you all that's needed to support both host and remote!

It lets you access and run an unattended IBM PC/XT/AT/PS2 or compatible from any remote location. On any personal computer. Day or night.

And with our new IRMA interface you can access a mini or mainframe, too.

This makes it ideally suited to remote customer service and technical support.

Because you can identify and solve problems from your office PC! Saving valuable time and the expense of a field service call.

With pcAnywhere you can compress and then transfer a file from your client's PC to yours, fix it, then transfer the file back to the client.

Virtually any program on the client's PC can be controlled from your keyboard. As though you were sitting in front of the client's computer.

Take remote control then give it back to your client's PC. Or keep both keyboards active at the same time and type messages back and forth.

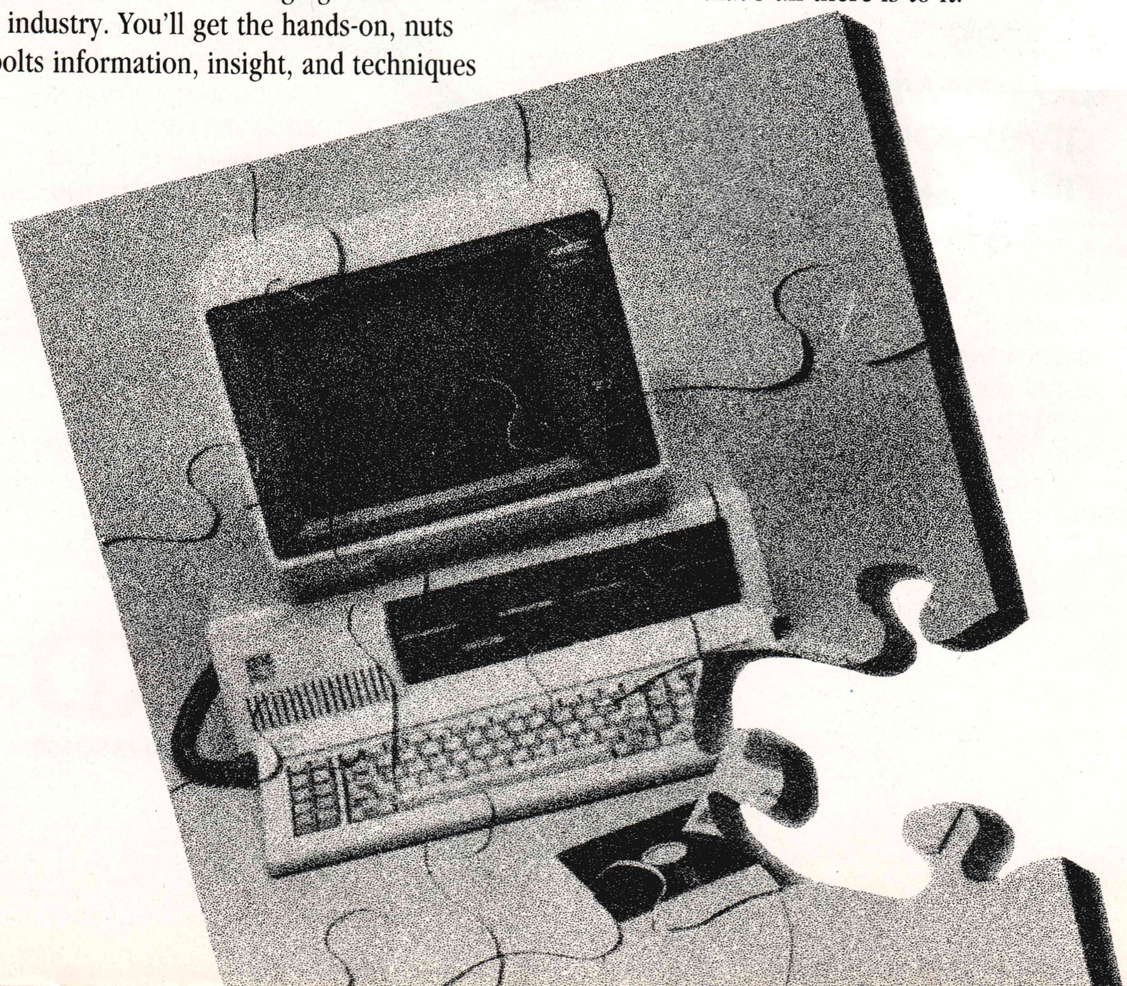
Call today for more information or to place an order. You'll find yourself in remote control before you know it!

pcAnywhere is a trademark of DMA.

EKD

764 Middle Country Road, P.O. Box Y, Selden, NY 11784 • **(516) 736-0500**

CIRCLE NO. 133 ON READER SERVICE CARD



47% SAVINGS

Yes! I want to subscribe to **Micro/Systems Journal**™ and save over 47% off the cover price.

☐ 1 Year (12 issues) $29.97 ☐ 2 Years (24 issues) $56.97

Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment Enclosed ☐ Bill me later

Card # ____________ Exp. date ____________

Signature ____________

Name ____________

Address ____________

City ____________ State ________ Zip ________

Savings based on a full one-year cover price of $47.40. Canada and Mexico add $6 for surface mail, $14 for airmail per year. All countries add $24 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

2004

A PUBLICATION OF M&T PUBLISHING, INC.

47% SAVINGS

Yes! I want to subscribe to **Micro/Systems Journal**™ and save over 47% off the cover price.

☐ 1 Year (12 issues) $29.97 ☐ 2 Years (24 issues) $56.97

Please charge my: ☐ Visa ☐ Master Card ☐ American Express

☐ Payment Enclosed ☐ Bill me later

Card # ____________ Exp. date ____________

Signature ____________

Name ____________

Address ____________

City ____________ State ________ Zip ________

Savings based on a full one-year cover price of $47.40. Canada and Mexico add $6 for surface mail, $14 for airmail per year. All countries add $24 for airmail per year. All foreign subscriptions must be prepaid in U.S. dollars drawn on a U.S. bank. Please allow 6-8 weeks for delivery of first issue.

2004

A PUBLICATION OF M&T PUBLISHING, INC.

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

Micro/Systems Journal™
Box 3713
Escondido, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

Micro/Systems Journal™
Box 3713
Escondidio, CA 92025-9843

No Postage
Necessary
If Mailed
In The
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 790, REDWOOD CITY, CA

Postage Will Be Paid By Addressee

Micro/Systems Journal™
Box 3713
Escondidio, CA 92025-9843

Listing One *(Listing continued, text begins on page 90.)*

```
void    initializeManagers()

            {
            /* local variable */
            Handle          someDay ;

            /* get some space */
            MoreMasters() ;
            /* get some master pointers */
            if (someDay = NewHandle(humungousBlock))    /* grow a maximal heap by */
                   DisposHandle (someDay) ;             /* asking for the future */
                                                        /* get those managers going */

            InitGraf(&thePort) ;                        /* set up  Quickdraw */

            InitFonts();                                /* set  up the Font Manager */

            InitWindows();
            /* set up the Window Manager */
            InitMenus();                                /* set  up the Menu Manager */

            TEInit();
            /* set up Text Edit */
            InitDialogs (noResumeProcedure) ;           /* set up the Dialog Manager */

            /* final adjustments */
            FlushEvents (everyEvent, dontStop ) ;       /* clear the event queue */
            InitCursor();
            /* turn the cursor on */
            }

/*---------------------------- studyAndSetEnvironment -------------------------*/

/*   check out screens, machines, ROMs, et al */

void    studyAndSetEnvironment ()

            {
            /* check out the screen */
            screenRect = screenBits.bounds ;
            screenHeight = screenRect.bottom - screenRect.top ;
            screenWidth = screenRect.right - screenRect.left ;

            /* determine height of the menu bar */
            if      ( ROM85 & 0x8000 )
                   menuBarHeight = stdMBarHeight ;      /* for 64K ROMs        */
            else
                   menuBarHeight = MBarHeight ;         /* for newer  ROMs     */

            }


/*-------------------------------- getThatDialogCookin -------------------------*/

/* set up and draw our main modal dialog window */

void    getThatDialogCookin ()

            {
            /* local variables */
            Point                           tempPoint ;
            Rect                            scratch ;
            ControlHandle  theButton ;

            /* get the dialog window */
            ourDialog = GetNewDialog (ourDialogID, storeInHeap, inFront) ;

            /* adjust its position  */
            MoveWindow ( ourDialog,
                       (tempPoint = figureCenteredRectTLC  (&(*ourDialog).portRect)).h,
                        tempPoint.v, inFront ) ;


            /* make dialog window the current grafPort so we can change its font */
            SetPort (ourDialog) ;

            /* change its font to Geneva 12 */
            TextFont (geneva) ;
            TextSize (12) ;

            /* show the dialog */
            ShowWindow (ourDialog ) ;
            }


/*-------------------------------- dealWithDialogItem -------------------------*/

 /* deal with the hit item */

void    dealWithDialogItem (theItem)
            int             theItem ;

            {
```

(continued on next page)

TO THE MACS

Listing One
(Listing continued, text begins on page 90.)

```
/* local constants */
#define              oolSize                 6

/* local variables */
static short  onOffList[oolSize] = {        orwellItem, hupCoupleItem,

              ronItem, saveAsItem,

              pinheadItem, duplicateItem} ;
/* case out on the item */
switch (theItem)
       {
       case quitItem:
              finished = true ;
              break ;
       case orwellItem:
              doOrwellItem () ;
              break ;
       case snapshotItem:
              doSnapshotItem () ;
              break ;
       case mushroomItem:
              doMushroomItem () ;
              break ;
       case openItem:
              doOpenItem () ;
              break ;
       case saveAsItem:
              doSaveAsItem () ;
              break ;
       case flipItem:
              doFlipItem () ;
              break ;
       case someOffItem:
              doSomeOffItem (onOffList, oolSize) ;
              break ;
       case someOnItem:
              doSomeOnItem (onOffList, oolSize) ;
              break ;
       case copyrightItem:
              doCopyrightItem () ;
              break ;
       default:
```

```
                        break ;
                }

            /* remove local constants */
            #undef        oolSize
            }


/*--------------------------------- doOrwellItem -----------------------------*/

/* deal with a click of the orwellItem  button */

void   doOrwellItem ()

            {
            /* local constants */
            #define                     cyclesDesired        4
            #define                     delayTicksOne        20
            #define                     delayTicksTwo        10

            /* local variables */
            Rect                        scratch ;
            ControlHandle  theItemHandle ;
            short                       cycleCounter ;
            ControlHandle  ronItemHandle ;

            /* get a handle to the button */
            GetDItem ( ourDialog, orwellItem, &scratch, &theItemHandle, &scratch) ;

            /* hilite the button */
            HiliteControl (theItemHandle, hilitedHS ) ;

            /* get a handle to the ronItem button */
            GetDItem ( ourDialog, ronItem, &scratch, &ronItemHandle, &scratch) ;

            /* run several fade cycles on the ronItem button  */
            for ( cycleCounter = 0; cycleCounter < cyclesDesired; cycleCounter++)
                  {
                  /* fade out */
                  HiliteControl (ronItemHandle, inactiveHS ) ;

                  /* wait a while */
                  Delay (delayTicksOne, &scratch) ;

                  /* back into view */
                  HiliteControl (ronItemHandle, activeHS ) ;
```

(continued on next page)

Listing One *(Listing continued, text begins on page 90.)*

```
                /* wait a while */
                Delay (delayTicksTwo, &scratch) ;
                }

        /* unhilite the button */
        HiliteControl (theItemHandle, activeHS ) ;

        /* remove local constants */
        #undef          cyclesDesired
        #undef          delayTicksOne
        #undef          delayTicksTwo
        }

/*--------------------------------- doSnapshotItem ----------------------------*/

/* deal with a click of the snapshotItem button */

void    doSnapshotItem ()

        {
        /* local variables */
        ControlHandle theItemHandle ;
        Rect                            scratch ;
        GrafPtr                         entryGrafPort ;

        /* get a handle to the button */
        GetDItem ( ourDialog, snapshotItem, &scratch, &theItemHandle, &scratch)
;

        /* hilite the button */
        HiliteControl (theItemHandle, hilitedHS ) ;

        /* save a pointer to the grafPort */
        GetPort(&entryGrafPort) ;

        /* this lets me take some snapshots */
        /* has no effect unless you have a desk accessory named Camera */

        OpenDeskAcc ("\007\000Camera") ;

        /* restore the grafPort */
        SetPort(entryGrafPort) ;

        /* unhilite the button */
        HiliteControl (theItemHandle, activeHS ) ;
        }

/*--------------------------------- doMushroomItem ----------------------------*/

/* deal with a click of the mushroomItem  button */

void    doMushroomItem ()

        {
        /* local constants */
        #define                         cyclesDesired           4
        #define                         delayTicks                      30

        /* local variables */
        Rect                            scratch ;
        short                           cycleCounter ;
        ControlHandle  itemHandleOne ;
        ControlHandle  itemHandleTwo ;

        /* get a handle to the button */
        GetDItem ( ourDialog, mushroomItem, &scratch, &itemHandleOne, &scratch)
;

        /* hilite the button */
        HiliteControl (itemHandleOne, hilitedHS ) ;

        /* get a handle to the bumperStickersItem button */
        GetDItem ( ourDialog, bumperStickersItem, &scratch,
                                        &itemHandleTwo, &scratch) ;

        /* run several fade cycles on the bumperStickersItem button  */
        for ( cycleCounter = 0; cycleCounter < cyclesDesired; cycleCounter++)
                {
                /* fade out */
                HiliteControl (itemHandleTwo, hilitedHS ) ;

                /* wait a while */
                Delay (delayTicks, &scratch) ;

                /* back into view */
                HiliteControl (itemHandleTwo, activeHS ) ;

                /* wait a while */
                Delay (delayTicks, &scratch) ;
                }

        /* unhilite the button */
        HiliteControl (itemHandleOne, activeHS ) ;
```

```
        /* remove local constants */
#undef          cyclesDesired
#undef          delayTicks
        }

/*---------------------------------- doOpenItem ------------------------------*/

/* deal with a click of the openItem button */

void    doOpenItem ()

        {
        /* local variables */
        Rect                        scratch ;
        ControlHandle  theItemHandle ;
        DialogTHndl          theDLOGHandle ;
        SFReply                     dummyReply ;

        /* get a handle to the button */
        GetDItem ( ourDialog, openItem, &scratch, &theItemHandle, &scratch) ;

        /* hilite the button */
        HiliteControl (theItemHandle, hilitedHS ) ;

        /* run the standard file open dialog */
        theDLOGHandle = (DialogTHndl) GetResource ('DLOG', getDlgID) ;
        SFGetFile (figureCenteredRectTLC (&(**theDLOGHandle).boundsRect),
                                    nil, nil, allTypes, nil, nil, &dummyReply ) ;


        /* unhilite the button */
        HiliteControl (theItemHandle, activeHS ) ;
        }

/*-------------------------------- doSaveAsItem -------------------------------*/

/* deal with a click of the saveAsItem button */

void    doSaveAsItem ()

        {
        /* local variables */
        Rect                        scratch ;
        ControlHandle  theItemHandle ;
        DialogTHndl          theDLOGHandle ;
        SFReply                     dummyReply ;

        /* get a handle to the button */
        GetDItem ( ourDialog, saveAsItem, &scratch, &theItemHandle, &scratch) ;

        /* hilite the button */
        HiliteControl (theItemHandle, hilitedHS ) ;

        /* run the standard file open dialog */
        theDLOGHandle = (DialogTHndl) GetResource ('DLOG', putDlgID) ;
        SFPutFile (figureCenteredRectTLC (&(**theDLOGHandle).boundsRect),
                        "\015Save file as:", "\021Current File Name",
                          nil, &dummyReply ) ;


        /* unhilite the button */
        HiliteControl (theItemHandle, activeHS ) ;
        }

/*-------------------------------- doFlipItem --------------------------------*/

/* deal with a click of the flipItem button */

void    doFlipItem ()

        {
        /* local constants */
#define              numButtons          9
#define              cyclesDesired 4
#define              delayTicks          15

        /* local variables */
        static short  flipList [numButtons] =
                          {    hupCoupleItem,          mouthOpensItem,
                          trashItem,                   melancholyItem,
                          eeekShrinkItem,              mushroomItem,
                          duplicateItem,               orwellItem,
                          bumperStickersItem
        } ;
        Rect                        scratch ;
        ControlHandle  theItemHandle ;
        short                       cycleCounter ;
        short                       index ;
        ControlHandle  tempItemHandle ;
```

(continued on next page)

Listing One *(Listing continued, text begins on page 90.)*

```
                    /* get a handle to the button */
                    GetDItem ( ourDialog, flipItem, &scratch, &theItemHandle, &scratch) ;

                    /* hilite the button */
                    HiliteControl (theItemHandle, hilitedHS ) ;

                    /* run several animation cycles on a group of content-changing buttons
*/
                    for ( cycleCounter = 0; cycleCounter < cyclesDesired; cycleCounter++)
                              {
                              /* hilite all the buttons in the group  */
                              for (index = 0 ; index < numButtons ; index++)
                                        {
                                        GetDItem ( ourDialog, flipList[index], &scratch,
                                                            &tempItemHandle, &scratch) ;
                                        HiliteControl (tempItemHandle, hilitedHS ) ;
                                        }

                              /* wait a while */
                              Delay (delayTicks, &scratch) ;

                              /* unhilite all the buttons in the group */
                              for (index = 0 ; index < numButtons ; index++)
                                        {
                                        GetDItem ( ourDialog, flipList[index], &scratch,
                                                            &tempItemHandle, &scratch) ;
                                        HiliteControl (tempItemHandle, activeHS ) ;
                                        }

                              /* wait a while */
                              Delay (delayTicks, &scratch) ;
                              }

                    /* unhilite the button */
                    HiliteControl (theItemHandle, activeHS ) ;

                    /* remove local constants */
                    #undef          numButtons
                    #undef          cyclesDesired
                    #undef          delayTicks
                    }

/*------------------------------ doSomeOffItem ------------------------------*/

/* deal with a click of the someOffItem button  */

void    doSomeOffItem (theOffList, listSize)
                    short           theOffList[] ;
                    short           listSize ;

                    {
                    /* local variables */
                    short                           index ;
                    Rect                            scratch ;
                    ControlHandle  theItemHandle ;
                    ControlHandle  tempItemHandle ;

                    /* get a handle to the button  */
                    GetDItem ( ourDialog, someOffItem, &scratch, &theItemHandle, &scratch) ;

                    /* hilite the button */
                    HiliteControl (theItemHandle, hilitedHS ) ;

                    /* for each item in the list */
                    for (index = 0 ; index < listSize ; index++)
                              {
                              /* get a handle to the item */
                              GetDItem ( ourDialog, theOffList[index], &scratch,
                                                  &tempItemHandle, &scratch) ;

                              /* inactivate the item */
                              HiliteControl (tempItemHandle, inactiveHS ) ;
                              }

                    /* unhilite the someOffItem button  */
                    HiliteControl (theItemHandle, activeHS ) ;
                    }

/*------------------------------ doSomeOnItem ------------------------------*/

/* deal with a hit of the someOnItem button  */

void    doSomeOnItem (theOnList, listSize)
                    short           theOnList[] ;
                    short           listSize ;

                    {
                    /* local variables */
                    short                           index ;
                    Rect                            scratch ;
                    ControlHandle  theItemHandle ;
                    ControlHandle  tempItemHandle ;
```

```
                /* get a handle to the button  */
                GetDItem ( ourDialog, someOnItem, &scratch, &theItemHandle, &scratch) ;

                /* hilite the button */
                HiliteControl (theItemHandle, hilitedHS ) ;

                /* for each item in the list */

                for (index = 0 ; index < listSize ; index++)
                        {
                        /* get a handle to the item */
                        GetDItem ( ourDialog, theOnList[index], &scratch,
                                              &tempItemHandle, &scratch) ;

                        /* activate the item */
                        HiliteControl (tempItemHandle, activeHS ) ;
                        }

                /* unhilite the someOnItem button  */
                HiliteControl (theItemHandle, activeHS ) ;
                }

/*--------------------------------doCopyrightItem ----------------------------*/

/* deal with a hit on the copyrightItem button */

void    doCopyrightItem ()

                {
                /* local variables */
                Rect                            scratch ;
                ControlHandle  theItemHandle ;
                DialogPtr               copyrightDlog ;
                Point                           tempPoint ;

                /* get a handle to the button */
                GetDItem ( ourDialog, copyrightItem, &scratch, &theItemHandle, &scratch)
;

                /* hilite the button */
                HiliteControl (theItemHandle, hilitedHS ) ;

                /* pull in the copyright notice modal dialog */
                copyrightDlog = GetNewDialog (copyrightDlogID, storeInHeap, inFront) ;

                /* center it on the screen */
                MoveWindow ( copyrightDlog,
                             (tempPoint = figureCenteredRectTLC
(&(*copyrightDlog).portRect)).h,
                              tempPoint.v, inFront ) ;

                /* show the copyright notice */
                ShowWindow (copyrightDlog ) ;

                /* wait until the user clicks the mouse in the dialog */
                ModalDialog (noFilterProcedure, &scratch) ;

                /* get rid of the dialog */
                DisposDialog (copyrightDlog) ;

                /* unhilite the button */
                HiliteControl (theItemHandle, activeHS ) ;
                }

/*--------------------------- figureCenteredRectTLC ---------------------------*/

/* given a rectangle, returns the top left corner  position that will
       center the rectangle inside screen area that's below the menu bar */

Point   figureCenteredRectTLC (theRect)




                Rect    *theRect ;

                {
                 /* local variable */
                Point   theResult ;

                 /* figure the vertical position */
                 theResult.v = menuBarHeight + ((screenHeight - menuBarHeight) -
                                                                        (theRect-
>bottom - theRect->top) )  /  2 ;

                 /* figure the horizontal position */
                 theResult.h = ( screenWidth - (theRect->right - theRect->left) )   /
2 ;
```

(continued on next page)

Listing One *(Listing continued, text begins on page 90.)*

```
   /* done, so return the point */
   return (theResult) ;
  }
```

End Listing One

Listing Two

```
/*------------------------------ file information -----------------------------*/

/*
        custom controls demo.h

        private definitions for custom controls demo.c

        edited and compiled with Lightspeed C 2.13

        ©1987 by Stan Krute   --    all rights reserved

        timestamp:          5:56 pm PST                    November 16, 1987
        spacestamp:         18617 Camp Creek Road       Hornbrook, California 96044


        this file looks good in 9 point Courier, LSC tabs set to 3
*/


/*-------------------------------- constants ----------------------------------*/

/* booleans */
#define              true                         1
#define              false                        0

/* parameters */
#define              humungousBlock               0x8FFFFFFF
#define              noResumeProcedure            0
#define              dontStop                     0
#define              nil                          0
#define              allTypes                     -1

/* control stuff */
#define              activeHS                     0              /* three hilite states */
#define              inactiveHS                   255
#define              hilitedHS                    10


/* menu stuff */
#define              stdMBarHeight  20
#define              titleMenuID                  1
#define              append                       0

/* dialog stuff */
#define         storeInHeap                       0
#define              inFront                      -1
#define              ourDialogID                  1
#define              copyrightDlogID              210
#define              noFilterProcedure            0

#define              quitItem                     1
#define              orwellItem                   2
#define              snapshotItem                 3
#define              bumperStickersItem           4
#define              mushroomItem                 5

#define              hupCoupleItem                6
#define              openItem                     7
#define              ronItem                      8
#define              saveAsItem                   9
#define              duplicateItem                10

#define              woozyItem                    11
#define              trashItem                    12
#define              flipItem                     13
#define              mouthOpensItem               14
#define              copyItem                     15

#define              melancholyItem               16
#define              pinheadItem                  17
#define              someOffItem                  18
#define              someOnItem                   19
#define              eeekShrinkItem               20
```

```
#define              copyrightItem        21

/*-------------------------------- type definitions --------------------------*/

typedef              short  boolean;

/*--------------------------- function prototypes ----------------------------*/

void    main (void) ;
void    initializeManagers (void) ;
void    studyAndSetEnvironment (void) ;
void    getThatDialogCookin (void) ;
void    dealWithDialogItem (int   theItem) ;
void    doSnapshotItem (void) ;
void    doOrwellItem (void) ;
void    doMushroomItem (void) ;
void    doOpenItem (void) ;
void    doSaveAsItem (void) ;
void    doFlipItem (void) ;
void    doSomeOffItem (short  theOffList[], short   listSize);
void    doSomeOnItem (short  theOnList[], short   listSize);
void    doCopyrightItem (void) ;
Point   figureCenteredRectTLC (Rect    *theRect) ;

/*----------------------------- global variables ----------------------------*/

boolean                    finished ;               /* indicates when the
program should end */
DialogPtr            ourDialog ;              /* points to our main dialog */

int                        menuBarHeight ;      /* know your environment... */
int                        screenHeight ;
int                        screenWidth ;
Rect                       screenRect ;
```

End Listings

Listing One *(Listing continued,* *text begins on page 108.)*

```
Listing One.  Example for declaring various objects that are
              variations of stack structures
PROGRAM Test_Objects(input,output);

{ Simple example for objects using TML Pascal running on a Mac Plus }

TYPE RealStackReg = ARRAY [1..4] OF real;
     IntStackReg  = ARRAY [1..4] OF integer;

     { define common stack-related variables and routines }
     TStack = OBJECT
        height : integer;
        ErrorFlag : boolean;
        PROCEDURE TStack.IStack;
        PROCEDURE TStack.Inc(VAR i : integer);
        PROCEDURE TStack.Dec(VAR i : integer);
     END;

     { define a 4-register real-typed stack }
     TRealStack = OBJECT(TStack)
        Regs : RealStackReg;
        PROCEDURE Push(Item : real);
        FUNCTION Pop : real;
        PROCEDURE Add;
     END;

     { define a 4-register real-typed stack with automatic
     LastX register }
     THPStack = OBJECT(TRealStack)
        LastX : real;
        PROCEDURE Add; OVERRIDE;
     END;

     { define a 4-register integer-typed stack }
     TIntStack = OBJECT(TStack)
        Regs : IntStackReg;
        PROCEDURE Push(Item : integer);
        FUNCTION Pop : integer;
        PROCEDURE Add;
     END;


PROCEDURE TStack.IStack;
{ Initialize TStack objects by setting the Stack height to zero }
BEGIN
  height := 0
END; { Stack.IStack }
PROCEDURE TStack.Inc(VAR i : integer);

BEGIN
  i := i + 1;
END;

PROCEDURE TStack.Dec(VAR i : integer);

BEGIN
  i := i - 1;
END;


PROCEDURE TRealStack.Push(Item : real);

VAR i : integer;

BEGIN
   Inc(height);
   FOR i := 4 DOWNTO 2 DO
      Regs[i] := Regs[i-1];

   Regs[1] := Item;
END; { RealStack.Push }
FUNCTION TRealStack.Pop : real;

VAR i : integer;
BEGIN
   IF height > 0 THEN BEGIN
      Pop := Regs[1];
      FOR i := 1 TO 3 DO
```

```
        Regs[i] := Regs[i+1];
      Dec(height);
      ErrorFlag := FALSE;
    END
    ELSE BEGIN
      Pop := 1.0E+30;
      ErrorFlag := TRUE
    END;

END; { TRealStack.Pop }
PROCEDURE TRealStack.Add;

VAR i : integer;

BEGIN
  Regs[1] := Regs[1] + Regs[2];
  FOR i := 2 TO 3 DO
    Regs[i] := Regs[i+1];
END; { TRealStack.Add }

PROCEDURE THPStack.Add;

VAR i : integer;

BEGIN
  LastX := Regs[1];
  Regs[1] := Regs[1] + Regs[2];
  FOR i := 2 TO 3 DO
    Regs[i] := Regs[i+1];
END;{ THPStack.Add }

PROCEDURE TIntStack.Push(Item : integer);

VAR i : integer;

BEGIN
   Inc(height);
   FOR i := 4 DOWNTO 2 DO
      Regs[i] := Regs[i-1];

   Regs[1] := Item;
END; { TIntStack.Push }


FUNCTION TIntStack.Pop : integer;

VAR i : integer;

BEGIN
   IF height > 0 THEN BEGIN
      Pop := Regs[1];
      FOR i := 1 TO 3 DO
         Regs[i] := Regs[i+1];
      Dec(height);
      ErrorFlag := FALSE
   END
   ELSE BEGIN
     Pop := -32767;
     ErrorFlag := TRUE
   END;
END; { TIntStack.Pop }

PROCEDURE TIntStack.Add;

VAR i : integer;

BEGIN
  Regs[1] := Regs[1] + Regs[2];
  FOR i := 2 TO 3 DO
    Regs[i] := Regs[i+1];
END; { TIntStack.Add }

{ --------------- declarations of variables ------------------ }

VAR RealStack : TRealStack;
    IntStack : TIntStack;
    { HPStack : THPStack; }
    X : real;
    I : integer;
    ch : char;
```

(continued on next page)

Listing One *(Listing continued, text begins on page 108.)*

```
BEGIN
  NEW(RealStack);
  NEW(IntStack);
  NEW(HPStack);
  { exercise real-type stack }
  RealStack.IStack;
  RealStack.Push(1.0);
  RealStack.Push(2.0);
  RealStack.Push(3.0);
  RealStack.Push(4.0);
  RealStack.Add;
  X := RealStack.Pop;
  WRITELN('X = ',X); WRITELN;
  { use HPStack }
  HPStack.IStack;
  HPStack.Push(0.0);
  HPStack.Push(0.0);
  HPStack.Push(3.0);
  HPStack.Push(4.0);
  INHERITED HPStack.Add;
  X := HPStack.Pop;
  WRITELN('X = ',X);
  WRITELN('LastX = ',HPStack.LastX);
  WRITELN;
  { exercise integer-type stack }
  IntStack.IStack;
  IntStack.Push(1);
  IntStack.Push(2);
  IntStack.Push(3);
  IntStack.Push(4);
  IntStack.Add;
  I := IntStack.Pop;
  WRITELN('I = ',I); WRITELN;
  readln(ch);
END.
```

End Listing

THE PROGRAMMER'S SHOP

provides complete information, advice, guarantees and every product for Microcomputer Programming.

EXIM TOOLKIT:

ESSENTIAL BASIC PROGRAMMING TOOLS

A **must** for all BASIC programmers who use Microsoft's QuickBASIC or IBM's PC BASIC compilers, our feature-rich, easy-to-use **EXIM ToolKit** provides dozens of useful routines unavailable in BASIC or any other single software library sold today!

Here's just some of what the **EXIM ToolKit** can do:

With our **Screen Management** module, you get features like four-way scrolling, windowing and data entry/display in screen forms. You can also write to screen many times faster than with BASIC and develop software to copy areas of the screen for cut and paste functions.

The multi-user **Data Base Management** module lets multiple users access the same data files across a local area network and still maintain data integrity. Users can build, add and delete indexes; reformat files and much more. Now BASIC users can control data as never before.

The **Network Interface modules** make it possible for BASIC programs to issue NETBIOS commands.

BASIC limits you to 64K of RAM for strings. **EXIM ToolKit's Memory Manager** lets you access up to **256K additional RAM** for strings. And, we offer free technical support, 7 days a week.

Order your **EXIM ToolKit** today. It's only \$99.95, (\$199.95 with Network Interface Modules) plus \$5.00 shipping and handling. (No licensing or royalties required.) If within 30 days, you are not completely satisfied with the **EXIM ToolKit**, return it to your dealer for a complete refund.

The **EXIM ToolKit** is so comprehensive, we're convinced it's the only BASIC Software Library you'll ever need.

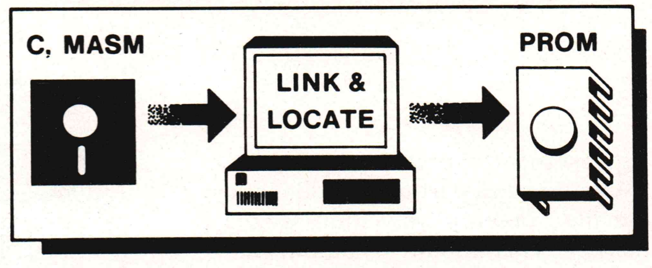
PS: \$ 45

EXIM

EXIM SERVICES OF N.A. INC.
P.O. Box 5417
Clinton, New Jersey 08809
(201) 735-7640

CIRCLE NO. 146 ON READER SERVICE CARD

FIRMWARE DEVELOPMENT



LINK & LOCATE enables PC users to produce ROM-based firmware for 8086/87/186 from object files generated by popular C compilers, such as from Microsoft, Lattice and Borland's Turbo C, and MASM from Microsoft, Provides full control of segment placement anywhere in memory. Supports output of Intel HEX file for PROM programmers, Intel OMF absolute object file for symbolic debuggers and in-circuit emulators. Includes Intel compatible linker, locator, librarian, hex formatter and cross reference generator. \$350.

NEW! Includes utility to support PC based PROM programmers.

SYSTEMS & SOFTWARE INC. **PS: \$329**

3303 Harbor Blvd., C11, Costa Mesa, CA 92626
Phone (714) 241-8650 FAX (714) 241-0377 TWX 910-695-0125

CIRCLE NO. 147 ON READER SERVICE CARD

Since 1981 HALO has been the industry standard library of graphic subroutines for the PC. HALO has the largest installed base of end-users and more ISV's than any graphics software environment.

Why? Because HALO grows with the industry. Graphics experts at Media Cybernetics are constantly improving HALO and expanding its compatibility. HALO supports 16 programming languages and over 125 devices. HALO is also compatible with IBM's new hardware series.

Media Cybernetics offers HALO programmers professional support, flexible, practical licensing terms, and the continuing commitment to assure that HALO will always be the most effective graphic toolkit in the industry.

List: \$300 **PS: \$209**

CIRCLE NO. 148 ON READER SERVICE CARD

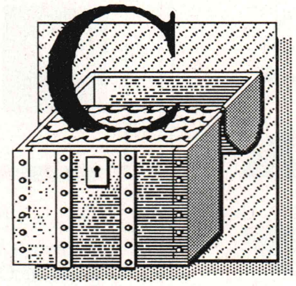
Call Today for FREE detailed information or try Risk-Free for 31 days, any product on this page.

HOURS: 8:30 A.M.-8:00 P.M. E.S.T.
5-D Pond Park Road, Hingham, MA 02043
Mass: 800-442-8070 or 617-740-2510

800-421-8006

THE PROGRAMMER'S SHOP
Your complete source for software, services and answers

A Preemptive Multitasking Kernel Continued, Lattice dBC, and Compiler Controversies



Last month I presented the users' guide and part of the code for a small, preemptive multitasking kernel. This month's column finishes up with the rest of the code and a description of how the subroutines work. The code appears in Listings One to Seven on pages 110–123 of the December 1987 issue.

I can only hope to give you a bare-bones introduction to operating system innards here. If you're starting out from scratch and want to pursue the matter further, I can recommend two excellent books. If you're interested in kernels only (which is often the case for ROM-based systems), Ted Biggerstaff's *System Software Tools* (Englewood Cliffs, N.J.: Prentice-Hall, 1986) presents both the underlying theory and the complete C source for a small multitasking kernel that runs on the IBM PC. Biggerstaff's OS includes a primitive screen I/O system, but hard stuff—such as the disk I/O—is relegated to DOS. Unlike my kernel, though, Biggerstaff's program lets you run other programs as subprocesses.

For a more in-depth introduction to operating systems in general, Andrew Tanenbaum's *Operating Systems: Design and Implementation* (Englewood Cliffs, N.J.: Prentice-Hall, 1987) is the best book on the subject that I've ever seen. It's both very complete in its coverage of the subject and extremely well written. (It's not often that a textbook is readable, but this book is a glowing exception

by Allen Holub

to the obfuscation-is-better rule.) Over the course of the book, Tanenbaum develops a complete Unix look-alike system, called MINIX, that runs on an IBM PC/XT or AT. He presents all the underlying theory in considerable depth as well as a complete implementation (in C) of the MINIX kernel. He covers virtually every aspect of operating system design, from the lowest-level disk driver (which interfaces directly to the hardware) up to the context-swapping code.

The book includes the full sources to the kernel, and if you spring for the accompanying disk (it's $80 from Prentice-Hall), you get an executable operating system and full sources for it and most of the other programs you need to actually use the operating system. (The disk contains a C compiler and an assembler, but you don't get the sources for these). The system provides something like 65 commands—the basic stuff such as cp, chmod, and so on and big stuff as well, such as a shell, an editor, grep, tar, uniq, roff, sort, pr, make, and ar. All these run under MINIX, of course, not DOS. But for $80 this is one of the deals of the century. Unlike Gnu, a public-domain Unix look-alike system, MINIX is not vaporware. In fact, the $80 you pay for MINIX is less than what you pay as a media fee when you get the "free" copy of Gnu.

Operating System Organization

As I mentioned last month, you can look at an interrupt-driven I/O system as an operating system. That is, each interrupt-service routine is a task, and the context swapping (changing from one task to another) is performed by the hardware every time an interrupt comes along. The task's priority is hard-wired into the hardware.

In most operating systems, however, context swaps are done in other ways: either a running task voluntarily yields (gives up control) to another task or a timer interrupt comes along and the running task is preempted (control is involuntarily taken from it). The part of the operating system that takes care of the swapping (and of figuring out which of several tasks should get control at any given moment) is called the kernel.

There are several ways to organize an operating system, two of which are shown in Figure 1, page 74. In the top diagram, the system is organized in onion-skin-like layers. Unix and MS-DOS are both examples of this organization. All I/O is done through the kernel, and access to the interrupts is done through the I/O system. The advantage of this approach is that it's easy to manage system resources—for example, you don't have to worry about two tasks both trying to write to the disk simultaneously; the kernel will act as a traffic cop.

The second diagram in Figure 1 shows how my own system is organized. This organization is mandated by the fact that I'm using DOS to do my I/O and that the kernel is part of the running program, not part of DOS itself. Resource management is a real problem in my system because DOS is not itself reentrant. It's up to the running task to assure that it cannot be preempted while it's performing a DOS call. You can do this in one of three ways.

The easiest method is to block all other tasks (the current task retains control of the CPU) using the *t_block()* and *t_release()* system calls. The problem here is that an I/O intensive task may use up an inordinate amount of system time.

An alternate approach is to use an exclusion semaphore. A semaphore is a length 1 message queue, created with a *t_makequeue()*

C CHEST

(continued from page 72)

system call. Normally a message is waiting at the queue. When a task wants a resource, it pends (waits for a message to arrive) at this queue by calling *t_wait( )*. When the task has finished with the resource, it posts the message back to the queue using *t_send( )*.

A third approach puts another level of isolation into the picture. Here, a special task is in charge of every resource. A second task sends a request to the resource task (again using the message mechanism), which performs an operation and then sends the result back to the calling task. For example, a single task might be in charge of all screen I/O. When a task wanted to send something to the screen, it would just post a message at the screen task's input queue (the message would probably be a pointer to a string) and the screen task would do the actual I/O. Another example is a disk I/O task. Here the requesting task would send a read-request message to the disk manager. This message would contain some sort of file descriptor, a count of the number of bytes to read, a pointer to a buffer, and the address of a queue to which the disk manager should send a response. The manager would fill the buffer and return a "done" message to the indicated return address when the disk access was complete.

The resource-manager approach is often the best for two reasons. First, messages can't possibly get intermingled because the screen task will process them in the order received. Second, the I/O process can be done in parallel with other tasks. For example, the disk manager could start up a DMA transfer and then suspend itself by waiting on a queue. The interrupt-service routine that's triggered by the DMA controller when the transfer is done posts a message to this same queue. In the interim, however, the disk manager is suspended, which means that other tasks can be active. Had you used the block/release strategy or an exclusion semaphore, all activity would stop while the active task waited for the transfer to complete. The kernel itself contains no resource managers, however. You'll have to write your own.

The Data Structures

So, what exactly is a task? Individual tasks (or processes) consist of several parts. First is the code itself. For reasons that will be apparent shortly, this code must be reentrant. That is, you must write it as if it were a recursive function—at least in terms of static-variable usage and so forth. If you're careful, several tasks can share the same code (because, like recursive subroutines, they all have unique stacks—more on this momentarily). The second part of a task is a task control block, or TCB. The TCB that's used by the kernel is defined in kernel.h (Listing One) starting on line 60. It's also shown in Example 1, page 78.

The TCB is shown graphically in Figure 2, below. The TCB structure forms a header, and the remainder of the structure is memory that will be used for a task's stack. The size of the stack is determined at task-create time (it's passed into *t_create( )* as an argument). TCBs (or pointers to them) are stored in one of three places: in an active list, which is a priority queue consisting of all tasks that have been preempted and are waiting to be activated; queued up at a message (I'll look at this process in a moment); or, if the task is the currently running task, a pointer to the task is held in the global variable *T_active*.

Several things happen when a timer interrupt comes along. First, the various message queues are examined, and if any of the tasks waiting at these queues have timed-out, they are added to the active list. Next, the priority of the running task is compared with the highest-priority task in the active list, and if the latter is higher, a context swap is

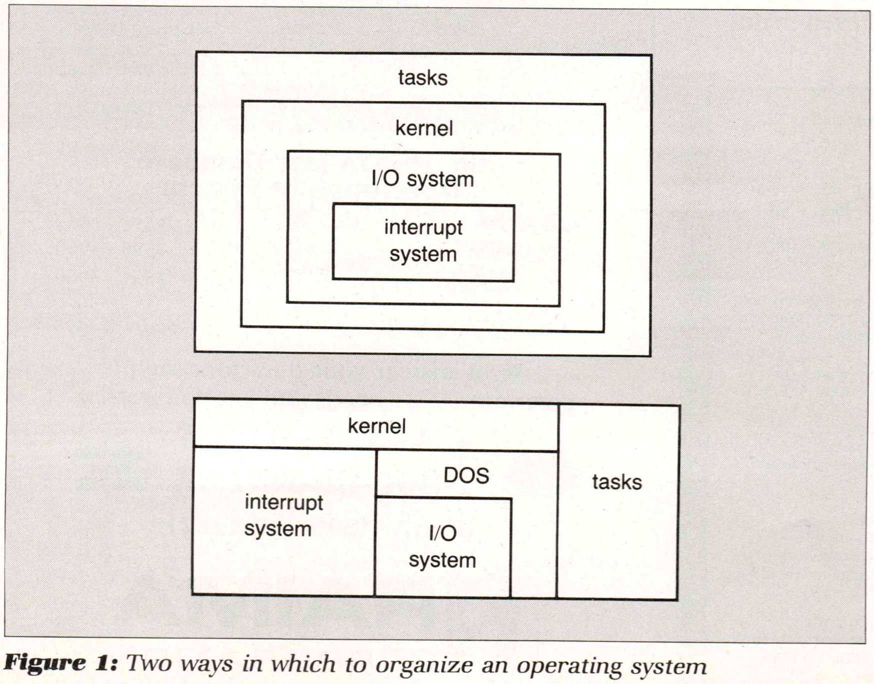


Figure 1: *Two ways in which to organize an operating system*

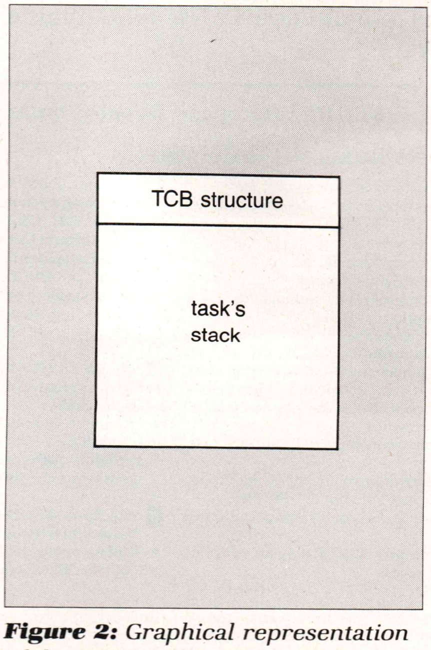


Figure 2: *Graphical representation of the TCB used by the kernel*

Simply the BEST C and Pascal on AT, 386, Sun, Apollo, RT, VAX, 370

"The most rock-solid C compiler in the industry. Superb technical support and portability. Superior code generated."
Gordon Eubanks, Symantec — Q&A (386).

"It simply works, with no trouble, no chasing strange bugs, and excellent warning and error messages ... a professional product."
Robert Lerche, Bay Partners.

"For large-scale software development, the highest quality C compiler available on the market today. Pragmas are great. Quality of support is exceptional." **Randy Neilsen, Ansa—Paradox (DOS,OS/2).**

"15% smaller and 15% faster than Lattice C."
Robert Wenig, Autodesk.

"Our software is running anywhere from 30 to 50% faster than when compiled under Lattice." **David Marcus, Micronetics.**

"We switched from Lattice due to a 10% reduction in code size. The compiler is very stable." **Lee Lorenzen, Ventura Software — Ventura Publisher, marketted by Xerox Corp.**

"Best quality emitted code by any compiler I've encountered. Often amazing." **Bill Ferguson, Fox Software — FoxBase (386).**

"Messages sometimes pointed out type mismatches, incorrect-length argument lists, and uninitialized variables that had been undetected for years [in 4.x bsd]." **Larry Breed, IBM ACIS [RT PC].**

"Diagnostics turned up bugs missed by other compilers. Rapid bug fixes by technical support, someone who knew what he was talking about. 80386 code is well optimized."
Tim Addison, Logistics Data Systems.

"386 protected mode support is fantastic, especially the access to large amounts of memory. It's mainframe compute power on a PC." **Dan Eggleston, Viewlogic.**

"The preprocessor supplied with Professional Pascal is quite useful. The code quality and control over segmentation and memory models are superior to MS Pascal." **Bob Wallace, QuickSoft.**

Check Out These Reviews

- **High C ™:**

Computer Language	February 1986, '87	
Dr. Dobb's Journal	August 1986	
PC Magazine	Jan. 27, 1987	(80386 version)
Dr. Dobb's Journal	July 1987	(80386 version)
BYTE Magazine	November 1987	(80386 version)

- **Professional Pascal ™:**

PC Magazine	Dec. 29, 1985	
Computer Language	May 1986	
PC Tech Journal	July 1986	
Journal of Pascal, Ada, & Modula-2	Nov.-Dec. 1986	
BYTE Magazine	Dec. '86, June '87	(80386 version)

Why MetaWare compilers

- They are specifically designed for serious software developers.
- They are reliable and robust: they don't break at every turn.
- Their generated code is the best, or near best, on each architecture.
- Their superior diagnostic messages help you produce better products more quickly.
- Your source can be ported with ease to the most popular systems.
- You can link mixed-language modules from our compilers, others
- You can benefit from high-level, personal technical support.
- You can take advantage of the latest ANSI C extensions, and/or extensive Pascal extensions. **High C** has been tracking the ANSI Standard for two years; **Professional Pascal** will soon have a VS Pascal compatibility switch and several Apollo Pascal ext'ns.
- You can take advantage of the latest 387 and Weitek 1167 support — we have the only compilers with Weitek real mode support.

Power Tools for Power Users

Ashton-Tate: dBase III Plus, MultiMate; Autodesk: AUTOCAD, AUTOSKETCH (8087, '387, Weitek); Boeing Computer Services (Sun); CASE Technology (Sun); CAD/CAM giant Daisy ('86, '386, VAX); Deloitte Haskins & Sells; Digital Research: FlexOS; GE; IBM: 4.3/RT, 4680 OS; Lifetree Software (Pascal): Volkswriter Deluxe, GEM-Write; Lugaru: Epsilon; NYU: Ada-Ed cmplr; Semantec: Q&A; Sky Computers; ... (Product names are trademarks of the companies indicated.)

Industrial-Strength

MetaWare C and Pascal compilers are designed for professional software developers. These tools are loaded with options to control them for special purposes. You can adjust the space-time trade-off in code quality. You can adjust external naming conventions to agree with linkers and operating systems. You can specify segment names for segmented architectures, and to help place code or data in particular places for embedded applications. You can select from five memory models for the 8086 family. And on and on.

A Partial List of Optimizations

Common subexpression and dead-code elimination, retention and reuse of register contents, jump-instruction size minimization, tail merging (cross jumping), constant folding, short-circuit evaluation of Boolean expressions, strength reductions, fast procedure calls, automatic mapping of variables to registers (where advantageous), ...

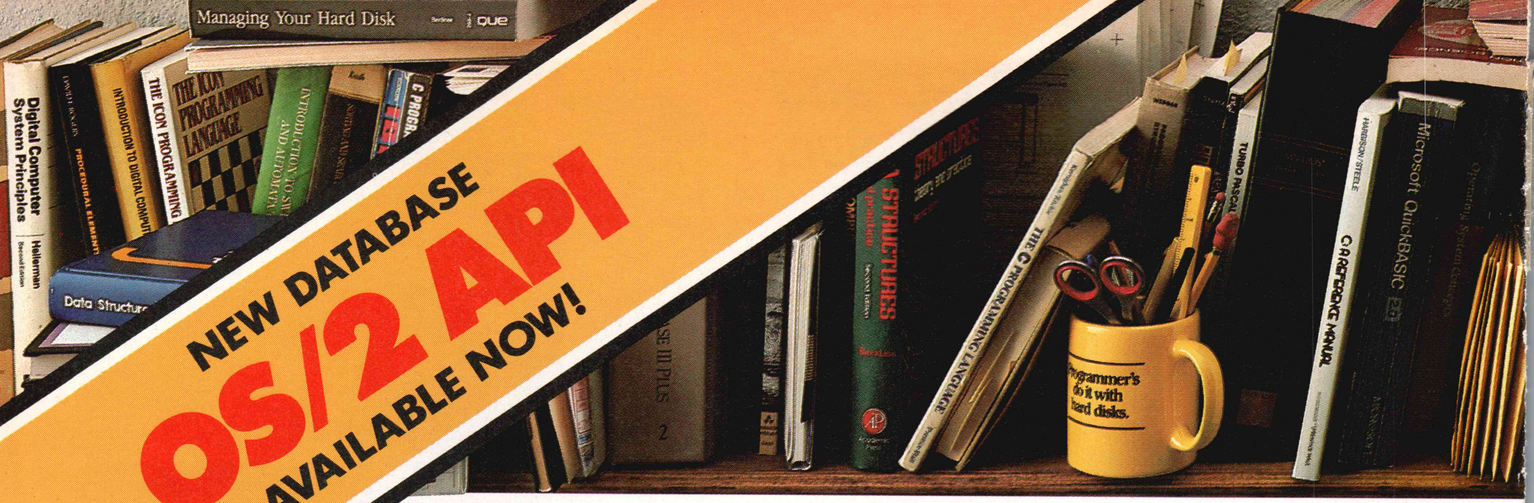
"Platform" — Code Quality

Sun,Apollo,SGI	— 18%, 3%, 26% > resident compiler (Dhrystone).
PC: DOS, OS/2	— 3-10% > Microsoft C; 30% > MS Pascal, LatticeC.
386 32-bit DOS	— no competitors, since November, 1986.
286, 386 UNIX	— 66% better than pcc (Dhrystone, 386).
VAX VMS	— ≈ DEC's excellent C and Pascal; better features.
VAX Ultrix	— 19% > pcc (Dhrystone); much > Berkeley Pascal.
RT PC/4.3bsd	— 89% > the original port of pcc (Dhrystone).
370 CMS,UNIX	— much better than any C, and VS Pascal.
AMD 29000	— >40,000 Dhrystones! Available in Q2, cross.

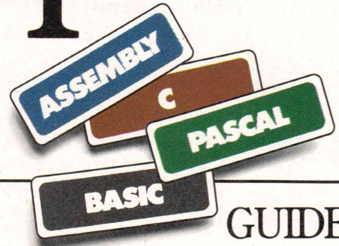
(408) 429-6382, telex 493-0879. ***Since 1979.***

Meta Ware™
INCORPORATED
903 Pacific Avenue, Suite 201 • Santa Cruz, CA 95060-4429

The Clear Choice for Large Programming Projects — PC Tech J.



Finally. A pro for people who h



Nobody ever said programming PCs was supposed to be easy.

But does it have to be tedious and time-consuming, too?

Not any more.

Not since the arrival of the remarkable new program in the lower right-hand corner.

Which is designed to save you most of the time you're currently spending searching through the books and manuals on the shelf above.

The Norton On-Line Programmer's Guides are a quartet of pop-up reference packages that do the same things in four different languages.

Each package consists of two parts: A memory-resident instant access program. And a comprehensive, cross-referenced database crammed with just about everything you need to know to program in your favorite language.

GUIDES DATA

Instant Access Program
- Memory-resident—uses just 71K.
- Full-screen or moveable half-screen view, with pull-down menus.
- Auto lookup and searching.
- Tools for compiling your own databases.

ASSEMBLY (600K of data)
- DOS Service Calls: All INT 21h services, interrupts, error codes, FCB and PSP fields, standard handles and more.
- ROM BIOS Calls: All ROM calls plus low RAM usage.
- Instruction Set: All 8088/86 instructions, addressing modes, flags, bytes per instruction, clock cycles and more.
- MASM: Pseudo-ops and assembler directives.
- Tables: ASCII chart, line-drawing charts, keyboard scan codes and more.

BASIC (270K each database)
- IBM BASICA, Microsoft QuickBASIC and TurboBASIC.
- Statements and Functions: Describes all statements and built-in library functions.
- Tables: Line-drawing characters, ASCII chart, keyboard codes, error codes, operators, etc.

C (600K each database)
- Microsoft C and Turbo C: Describes language, including statements, operators, data types and structures.
- Library Functions: Detailed descriptions of all functions, from abort () to write ().
- Preprocessor Directives: Describes commands, usage and syntax.
- Tables: ASCII chart, line-drawing characters, keyboard codes, error codes, operators, etc.

PASCAL—Turbo (360K of data)
- Language: Describes statements, syntax, operators, data types and records.
- Library: Describes the library procedures and functions.
- Tables: ASCII chart, line-drawing characters, keyboard codes, error codes, reserved words, etc.

(If you don't believe us, you might want to take a moment or two to examine the data box you just passed.)

You can, of course, find most of this

Designed for the IBM® PC, PC-AT and DOS compatibles. Available at most software

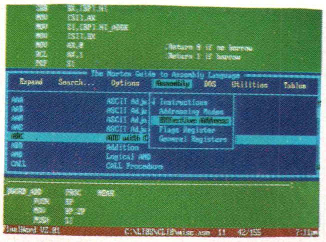
gramming tool
ate manual labor.

information in the books and manuals on our shelf.

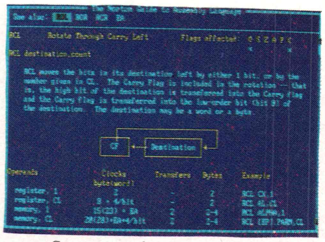
But Peter Norton—who's written a few books himself—figured you'd rather have it on your screen.

In seconds.

In either full-screen or moveable half-



A Guides reference summary screen (shown in blue) pops up on top of the program you're working on (shown in green).



Summary data expands on command into extensive detail. And you can select from a wide variety of information.

screen mode.

Popping up right next to your work. Right where you need it.

This, you're probably thinking, is precisely the kind of thinking that produced the classic Norton Utilities.

And you're right.

But even Peter Norton can't think of everything.

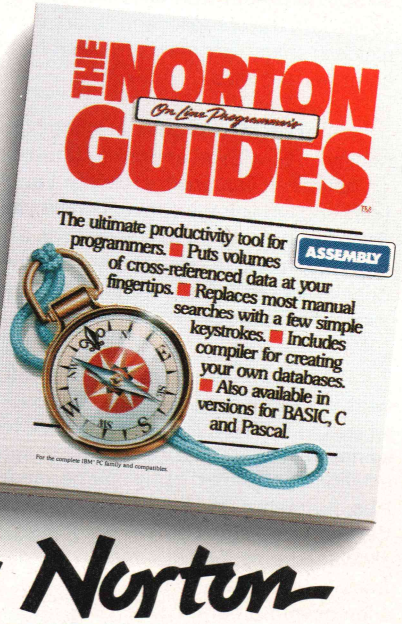
Which is why there's a built-in compiler for creating databases of your own.

And why all Guides databases are compatible with the instant access program in your original package.

So you can add more languages without spending a lot more money.

To get more information, call your dealer. Or call Peter Norton at 1-800-451-0303 Ext. 40.

And ask for some guidance.



Peter Norton
COMPUTING

dealers, or direct from Peter Norton Computing, Inc., 2210 Wilshire Blvd. #186, Santa Monica, CA 90403. 213-453-2361. Fax 213-453-6398, MCI Mail: PNCI ©1987 Peter Norton Computing

CIRCLE NO. 150 ON READER SERVICE CARD

Only WATERLOO C's programming environment delivers all 10 of the most wanted user features:

- ☑ High-speed compilation
- ☑ High-performance code
- ☑ Reentrant code options
- ☑ OS linkage options
- ☑ In-line functions
- ☑ ANSI compiler and library
- ☑ Full-screen library
- ☑ Full-screen symbolic debugger
- ☑ Extensive on-line documentation
- ☑ Development tools

WATERLOO C Version 3.0 is the high-performance development system for IBM 370 mainframe architectures under VM/CMS consisting of:

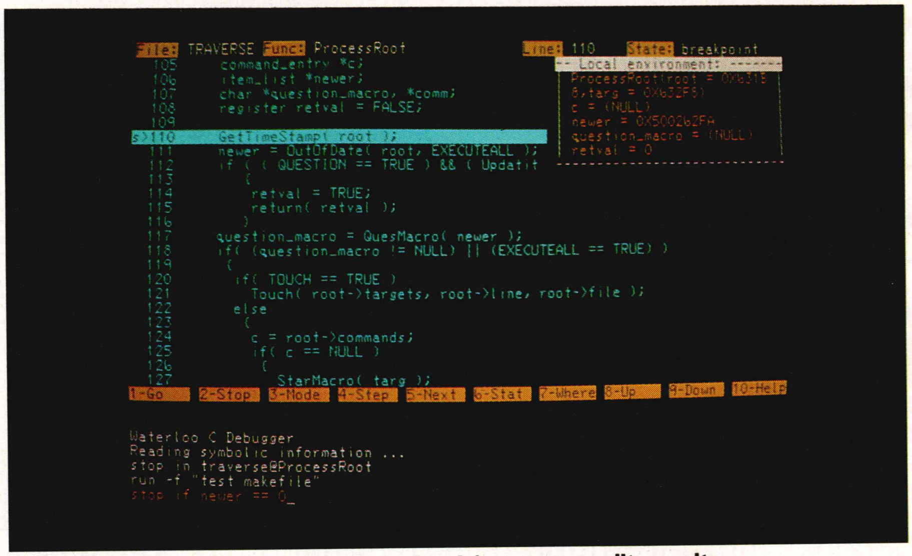
- **Optimizing C Compiler**
- **C Run-time Library**
- **Interactive Source-level Debugger**
- **Development Tools**

WATERLOO C is ideal for new software development and for porting existing software to IBM mainframes. It provides ANSI standard support in an integrated CMS environment for portability and efficiency. And it puts all the resources of a full development system at your fingertips so you can quickly achieve quality results.

WATERLOO C improves your response time and conserves your mainframe resources with an extremely fast compilation rate. A single phase from source to object means less overhead, while both the compiler and library can reside in shared segments for even faster compiles and links.

WATERLOO C generates the highest quality code with an exclusive state-of-the-art optimizer. Options for reentrant applications and OS linkage conventions mean flexibility in development. Moreover, a convenient option allows you to see the original source lines as comments in the assembler output.

The source-level debugger lets you work with your programs the way you wrote them. The full-screen interface feels so natural that it is easy to use. Debugging activity is performed using source variables, functions, and lines. And the execution environment allows stepping, breakpoints, tracing, and display and alteration of your program data.

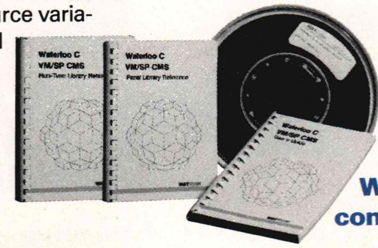


WATERLOO C 3.0 full-screen debugger: A quick route to quality results.

Completing the development environment are these important productivity tools:

- Update your programs with a single command: **MAKE.**
- Simplify multi-file maintenance with **GREP** and **DIFF**.
- Track source variables and functions with cross-reference utilities.
- Spot performance bottlenecks with a program execution profiler.

A full support framework provides complete technical backup. Access documentation easily on-line or in comprehensive reference manuals. Our Customer Support Center provides you with expert technical assistance. Newsletters keep you informed about product developments. And on-site installation and training workshops are available.



Discover the high-performance advantage of WATERLOO C. Write or call for our complete information package:

WATCOM

Find out how quickly and easily you can generate the highest quality code for IBM mainframes by writing or calling for the WATERLOO C complete product information package.

☐ Please send your product information package.

☐ I would like to arrange for an evaluation copy.

Name: ____________________

Title: ____________________

Company: ____________________

Street: ____________________

City: ____________________

State: ____________ Zip: ____________

Telephone: ____________________

WATCOM
Dept. DD-01
415 Phillip Street
Waterloo, Ontario, Canada N2L 3X2
Tel. (519) 886-3700

CIRCLE 201 ON READER SERVICE CARD

C CHEST
(continued from page 74)

performed as follows. The value of the system clock (held in *T_clock*) is copied into the *timestamp* field of the structure. Then all the registers except *SS* and *SP* are pushed onto the current task's stack. Next, the *SS* and *SP* registers are copied into the *ss* and *sp* fields of the TCB. Now the new task is dequeued from the active list and the old task is enqueued in its place. The new task is installed, first by copying the *ss* and *sp* fields into the corresponding registers (thereby making the new task's stack the active stack) and then by popping all the registers. If you've done a context swap, the popped registers will be those that were saved the last time the task was suspended, not the registers that were saved when you entered the timer interrupt-service routine (which are in the suspended task's TCB).

Note that the time of last preemption is considered when a task's priority is examined. That is, if two tasks have the same priority, then the one that has been waiting longest is considered the higher-priority task. This is how round-robin scheduling works.

The other fields in the TCB are used for debugging and message passing. The three debugging fields are *tag*, which points to the string that you passed into *t_create()*; *initial_sp*, which is the initial value of the stack pointer; and *status*, which tells you if a task is currently waiting for a message or not. The kernel doesn't use these fields for anything. The *tag* field is particularly useful because it lets you see which task is actually attached to a given TCB—you can pass the task name to *tcreate()*.

The remaining fields are used for the message-passing system. The *next* field is used to maintain a queue of tasks, all of which are waiting at the same message queue. The TCBs of these tasks are arranged as a linked list, with new tasks being added to the end of the list. The first TCB in the list gets a message when one arrives at the queue. This is accomplished by unlinking it from the list, setting the *msg* field to point

```
        typedef struct tcb
        {
            void            **sp;           /* Task-swap stuff */
            unsigned        ss;

            unsigned        priority;
            unsigned long   timestamp;

            unsigned        wait;           /* Message-management
stuff */
            struct tcb      *next;
            void            *msg;

            int             status;         /* debugging stuff */
            char            *tag;
            void            **initial_sp

            void            *stack[1];      /* Base of system stack
*/
        }
        TCB;
```

Example 1: *The TCB used by the kernel*

```
    typedef struct t_queue
    {
        int             signature;
        struct t_queue  *next;

        TCB             *task_h;        /* Task queue */
        TCB             *task_t;

        int             q_size;         /* Message queue */
        int             numele;
        void            **headp;
        void            **tailp;
        void            *queue[1];
    }
    T_QUEUE;
```

Example 2: *The* T_QUEUE *structure*

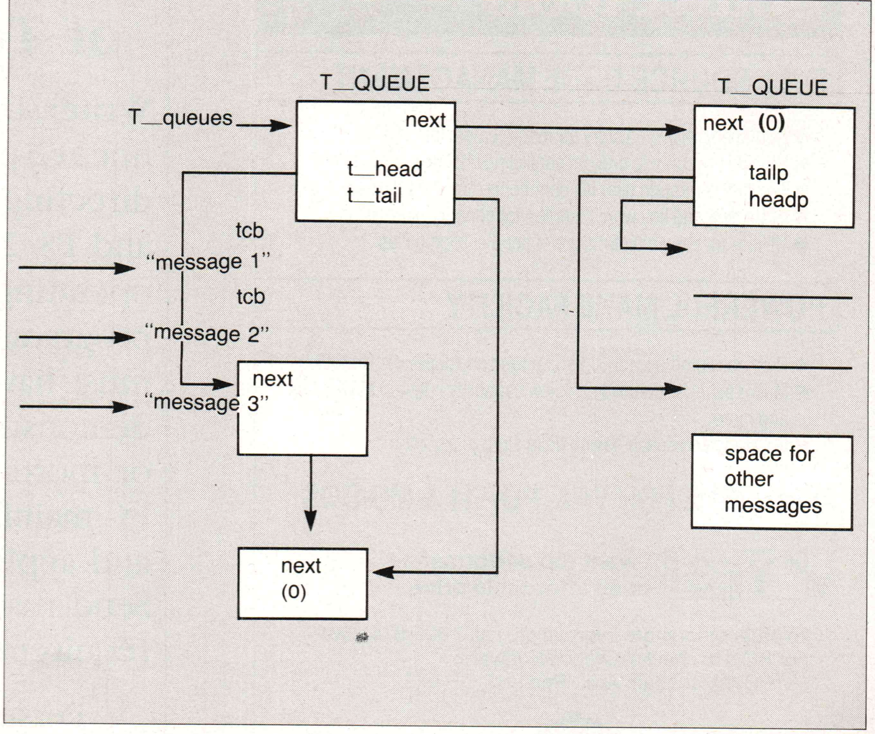


Figure 3: *A list of two queues, the first of which has two tasks waiting at it and the second of which has three messages enqueued and no tasks waiting. Only relevant fields are shown.*

C CHEST
(continued from page 79)

at the message, inserting the task into the active list, and then forcing a reschedule (as if a timer interrupt had come along). The *wait* field is a counting semaphore. It is initialized to the time-out value that you pass to *t_send()* and is decremented on every timer interrupt. When it gets to 0, the task is removed from the message queue and put back into the active list with the *msg* field set to *NULL*.

The next data structure of interest is the *T_QUEUE* structure, defined on lines 98–113 of Listing One and in Example 2, page 79. *T_QUEUE* is a variable-length structure similar to a TCB. Here, however, the area at the end of the queue is the message queue itself (rather than the stack). *Queue* is the first cell of that area. The *headp* and *tailp* fields point at the head and tail of the message queue and *numele* is the number of messages currently in the queue. The *q_size* field is the maximum number of messages that the queue can hold. Waiting tasks are queued up using the *task_h* and *task_t* fields. The former points to the first element in the linked list and the latter points at the last element.

Self-modifying code turned out to be an easy work-around

The queues themselves are also arranged in a linked list, put together in the order in which the queues are created. The timer interrupt-service routine chases down this second list looking for timed-out tasks. The *next* field points to the next queue in that list; the situation is illustrated in Figure 3, page 79. Finally, the *signature* field contains an arbitrary number that's used by some of the queue functions to make sure that the *T_QUEUE* pointer passed to them is valid.

The Subroutines

Once you understand the data structures, the actual code is pretty straightforward. Schedule.asm (Listing Two) contains the timer-interrupt-related stuff. It is very similar to the *speedup()* subroutine described in the September 1987 C Chest, so I won't go into details here.

_T_speedup() (line 182) speeds up the IBM PC system clock by an indicated factor and installs an interrupt-service routine that does context swaps when required; *_t_slowdown()* (line 241) slows the clock down again and removes the scheduler. *T_block()* and *t_release()* (lines 165–175) just set or clear a global flag (*blocked*, declared on line

C CODE FOR THE PC

source code, of course

C Source Code

Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
CQL Query System (SQL retrievals plus windows)	$325
Greenleaf Data Windows (windows, menus, data entry, interactive form design)	$315
Barcode Generator (specify Code 39 (alphanumeric), Interleaved 2 of 5 (numeric), or UPC)	$300
GraphiC 4.0 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$275
Vitamin C (MacWindows)	$200
resident C (TSRify C programs, DOS shared libraries)	$165
Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF)	$160
Greenleaf Functions (296 useful C functions, all DOS services)	$160
Essential C Utility Library (400 useful C functions)	$160
Essential Communications Library (C functions for RS-232-based communication systems)	$160
PC/IP (CMU/MIT TCP/IP implementation for PCs)	$100
B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
The Profiler (program execution profile tool)	$100
Entelekon C Function Library (screen, graphics, keyboard, string, printer, etc.)	$100
Entelekon Power Windows (menus, overlays, messages, alarms, file handling, etc.)	$100
QC88 C compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
CBTree (B+tree ISAM driver, multiple variable-length keys)	$80
MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
ME (programmer's editor with C-like macro language by Magma Software)	$75
Wendin PCNX Operating System Shell	$75
Wendin PCVMS Operating System Shell	$75
Wendin Operating System Construction Kit	$75
EZ_ASM (assembly language macros bridging C and MASM)	$60
Multi-User BBS (chat, mail, menus, sysop displays; uses Galacticomm modem card)	$50
Heap Expander (dynamic memory manager for expanded memory)	$50
Make (macros, all languages, built-in rules)	$50
Vector-to-Raster Conversion (stroke letters & Tektronix 4010 codes to bitmaps)	$50
Coder's Prolog (inference engine for use with C programs)	$45
PC/MPX (light-weight process manager; includes preemption and cooroutine packages)	$45
Biggerstaff's System Tools (multi-tasking window manager kit)	$40
CLIPS (rule-based expert system generator, Version 4.0)	$35
TELE Kernel (Ken Berry's multi-tasking kernel)	$30
TELE Windows (Ken Berry's window package)	$30
Clisp (Lisp interpreter with extensive internals documentation)	$30
Translate Rules to C (YACC-like function generator for rule-based systems)	$30
6-Pack of Editors (six public domain editors for use, study & hacking)	$30
ICON (string and list processing language, Version 6 and update)	$25
PTree (parse tree management)	$25
LEX (lexical analyzer generator)	$25
Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor)	$25
AutoTrace (program tracer and memory trasher catcher)	$25
C Compiler Torture Test (checks a C compiler against K & R)	$20
Benchmark Package (C compiler, PC hardware, and Unix system)	$20
TN3270 (remote login to IBM VM/CMS as a 3270 terminal on a 3274 controller)	$20
PKG (task-to-task protocol package)	$20
A68 (68000 cross-assembler)	$20
Xlisp 1.5a (Lisp interpreter including tiny-Prolog in Lisp)	$20
List-Pac (C functions for lists, stacks, and queues)	$20
XLT Macro Processor (general purpose text translator)	$20
C Tools (exception macros, wc, pp, roff, grep, printf, hash, declare, banner, Pascal-to-C)	$15

Data

DNA Sequences (GenBank 48.0 of 10,913 sequences with fast similarity search program)	$150
Protein Sequences (5,415 sequences, 1,302,966 residuals, with similarity search program)	$60
Webster's Second Dictionary (234,932 words)	$60
U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
USNO Floppy Almanac (high-precision moon, sun, planet & star positions)	$20
NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409
USA

Voice: (512) 258-0785
BBS: (512) 258-8831
FidoNet: 1:382/12
UUCP: uunet!acw!info

Free surface shipping on prepaid orders

MasterCard/VISA

C CHEST

(continued from page 80)

118) that will be examined by the interrupt-service routine, *serv*, on lines 281–345. The flag is checked on line 285, and if it's set, *numblk* is incremented (for statistics purposes only—it tells you how many interrupts were blocked) and control passes to *servexit* on line 327. The code on lines 328–240 vectors to the default DOS interrupt-service routine every *N* interrupts, where *N* is the original speedup factor; otherwise, a nonspecific EOI is issued to the timer chip and an *IRET* is executed.

The actual context swap is straightforward. Registers are pushed on lines 295–302. Note that the *CS*, *IP*, and flag registers are saved by the hardware as part of the interrupt-service procedure, so you don't have to push them again here. The current *SP* and *SS* are saved on lines 306 and 307, a local stack is then installed, and the C subroutine *_t_reschedule( )* is called on line 313. This subroutine (starting on line 512 of Listing Six) scans the list of messages and times out appropriate tasks (in the *for* loop on lines 532–550). It enqueues the running task's TCB and sets *T_active* to the new task's TCB when a context swap is required (on line 567).

Now control returns to the assembly-language code. The new TCB (which is the same as the old one if no swap is necessary) is fetched on line 315 of Listing One. Then the stack is restored, registers are popped, and you keep going. So, if you're doing a context swap, you'll save registers on the old TCB and then restore them from the new one. Note that the *CS*, *IP*, and flag registers are restored as part of the *IRET* instruction.

All the other assembly-language stuff is in swap.asm (Listing Three). The routines in this file are not particularly well structured—that is they save space by sharing code. One subroutine will jump into the middle of another. The central routine is *_t_swap_in( )* on lines 172–242. It is passed a pointer to a new TCB. It suspends the current task and installs a new one as if it were the timer interrupt-service routine. Again, registers are saved (lines 183–198), *T_active* is modified (line 216), and the new task is installed by popping registers (lines 227–242). Note that the subroutine's return address is used as the old task's saved *IP* register. This means that when control is restored to the task, you'll be back in the calling subroutine, just after the *_t_swap_in( )* call.

The other swap-related functions are *_t_install( )* (lines 142–168), which deletes the current task and installs a new one, and *_t_ shazam( )*, which starts up multitasking. Both of these jump into the middle of *_t_swap_in* (to the *shazam* label on line 223) to install the new task. *_T_shazam* also changes the stack probe subroutine, *chkstk*, which is called at the start of all normal subroutines. I couldn't use the normal *chkstk* because the task stacks are in strange places (in the middle of the heap in fact), so the default *chkstk* would always fail. The new routine checks the current

C CHEST
(continued from page 82)

SP against the current TCB's stack base and reports an error if the stack gets too large.

Note that the installation procedure involves actually modifying the first couple of instructions of the default *chkstk* subroutine to jump to the new one (on lines 271–274). I know this is a kludge, but Version 4.0 of the Microsoft compiler (and early betas of Version 5.0) made it virtually impossible to link in my own *chkstk*. (At least, I tried for a couple of hours with no success.) Self-modifying code turned out to be an easy work-around, but it's a stopgap measure, and I mean to fix it once I figure out how.

The final routine of interest is *__t__stop( )* (lines 94–138), which is called to terminate multitasking. It causes control to pass back to the instruction following the original *t__start* call—the return address was saved by *t__start( )*. *T__stop( )*'s argument is passed back as the return value of *t__start( )*. That is, a call to *t__stop(5)* will cause *t__start( )* to return a value of 5 to its caller. The situation is analogous to an *exit( )* call.

The remainder of the code is straightforward enough and sufficiently commented that there's no need to go through it here. I do suggest that you read it, though. It's pretty interesting. I'll finish with a caveat. I've used the routines printed here in a couple of application programs, and they seem to work fine. These program all use my own direct video I/O functions to write to the screen, however, and they don't do any disk I/O. It's possible to use DOS, but you'll have to implement one of the resource management strategies discussed earlier (or use a nonpreemptive system). Do not call DOS directly from a task without assuring that the task has control of the system for the life of the DOS call.

DBASE has spread like a disease through the small-business community.

I'm reasonably sure that the code presented here is bug free, but I won't swear to it. If you find a bug please report it to me c/o Software Engineering, P.O. Box 5679, Berkeley, CA 94705. Use E-mail to leave a message on CompuServe (because I don't log on reliably enough to guarantee that the message won't roll off the forum message board before I see it). Post a message on the forum, though, so everybody can see it.

Priority-Queue Routines in the Kernel

In addition to the various subroutines presented this and last month, the kernel uses the priority-queue routines from the June 1987 C Chest—you should look at that article for detailed information on how these routines work. I've modified the original code just a bit, however.

The *pq__look( )* routine now returns *NULL* if the queue is empty. It used to return garbage in this situation. I've also added a "replace" routine that replaces the highest-priority element in the queue with a new element, reheaps, and then returns the former top element. It's as if you did a delete of the old element followed by an insert of the new one, but it takes less time than would two operations. Both modifications are shown in Example 3. Also note that a bug fix that affects *pq__insert( )* was mentioned in the August 1987 C Chest, page 108.

```
void    *pq_look( queue )
PQ      *queue;
{
    return queue->nitems ?  queue->heap : NULL ;
}

/*----------------------------------------------*/

int  pq_replace( p, target, item )
PQ   *p;
void *item, *target;
{
    int slots_in_use;

    if( slots_in_use = p->nitems )
    {
        memcpy( target,  p->heap, p->itemsize );
        memcpy( p->heap, item,    p->itemsize );
        reheap_down( p, p->heap );
    }

    return slots_in_use ;
}
```

***Example 3:** The routines* pq__look( ) *and* pq__replace( )

Nifty Stuff: The Lattice dBC Library

Every programmer should have a database management library in his or her toolbox. This month I'll look at one such library—I'll look at others in future columns.

There are a host of database packages on the market, all with their own set of strengths and weaknesses. One of the weakest, from a database management point of view, is Ashton Tate's dBASE III. Unfortunately dBASE has spread like a disease throughout the small-business community. Consequently, it's often necessary to put together a dBASE-compatible application program, even if you'd rather use one of the more efficient database packages.

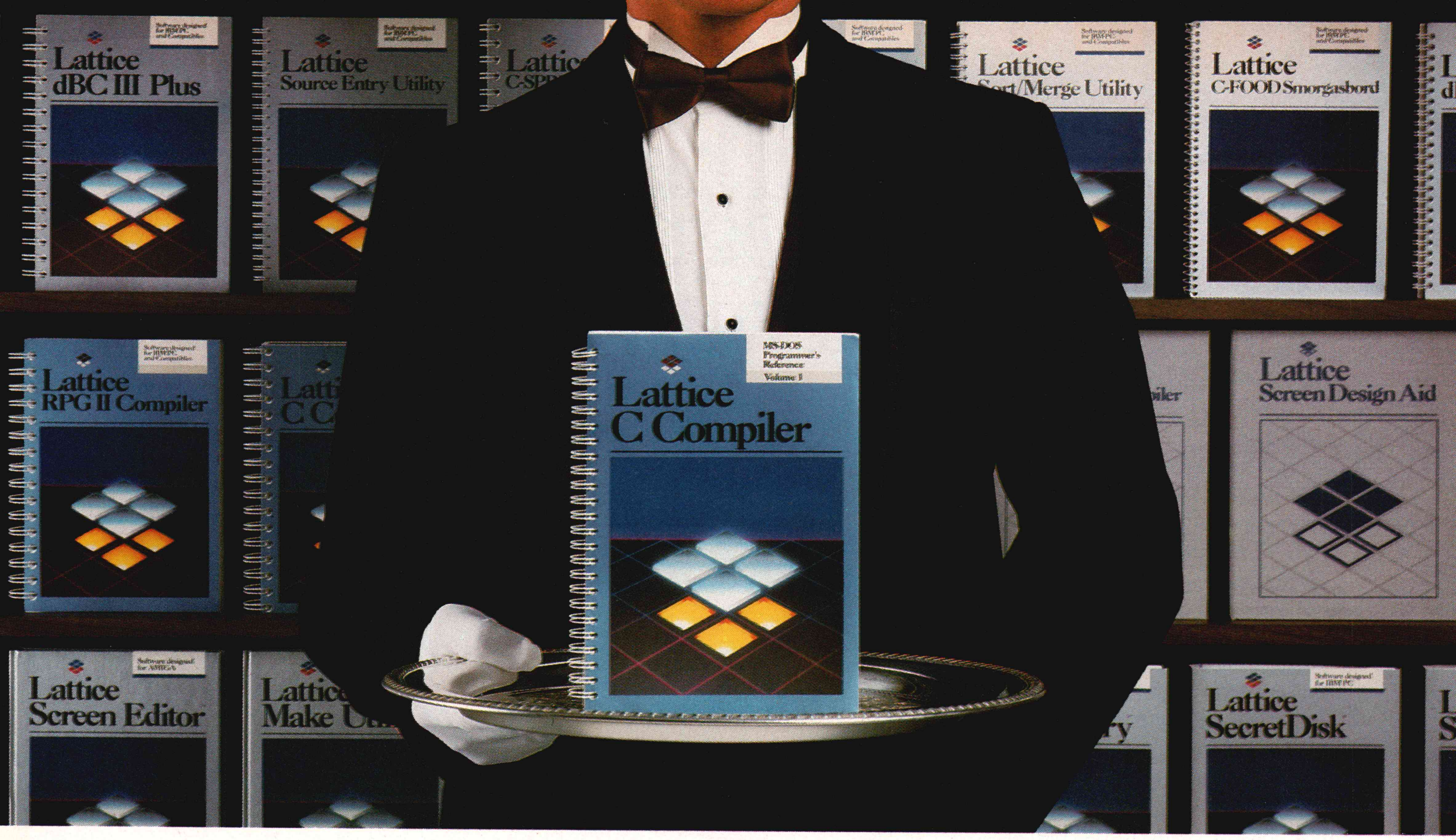
DBASE III is actually a programming language of sorts. In practice, it's so limited and hard to use that

no sane person would write a complicated application in it (at least not if they want to retain their sanity). As a consequence, several fourth-generation languages that support the dBASE file structure, such as FoxBASE and Quicksilver, have sprung up. These products have problems, too—they're limited in one way or another. I want to write my applications in C, not in some inherently inefficient and much too limited 4GL that's really designed for nonprogrammers. My programs will be faster and I can use all the other C functions (such as window management packages) that are at my disposal.

The Lattice dBC III and dBC III Plus packages provide a solution to this problem. These packages are subroutine libraries that you can link into your programs to access dBASE III or dBASE III PLUS databases. The version that I used links to Microsoft-generated code, not Lattice-generated (though versions for the Lattice compiler are available, of course). Versions of the package are available for versions of dBASE down to dBASE II. The main difference between the III and the III Plus version of dBC is networking support (you can lock whole databases, individual records, and bytes within the record). The package comes with small-, medium-, and large-model libraries as well as a demo program (and the sources for the demo program). Though the version that I used expected Microsoft C, Version 4.0, it linked into a program compiled under Version 5.0 without difficulty. (You can't use the new combined libraries, however—the components have to be on the disk.)

I have very mixed feelings about this package. On the plus side, it really does what's required. It took me an hour or so to read the manual, and a few hours later I had a working C program that was accessing a database created by dBASE III PLUS. You should have no difficulty doing the same, even if you've never worked with dBASE itself. Keep all this in mind as you read the negative comments that follow. Though the package has problems, most of them are fixable and dBC is really useful.

Unfortunately, this ease of use has



Our software comes with something no one else can offer.

Lattice Service.

When you join the Lattice family of customers, you'll discover that your software purchase is backed by more than just an excellent warranty. It's backed by unparalleled technical support. By a total commitment to your success and satisfaction. And by Lattice's dedication to excellence in products and services.

Unlike other software manufacturers who charge you for services after you've purchased their product, Lattice offers a unique package of support programs at a price we can all live with—FREE.

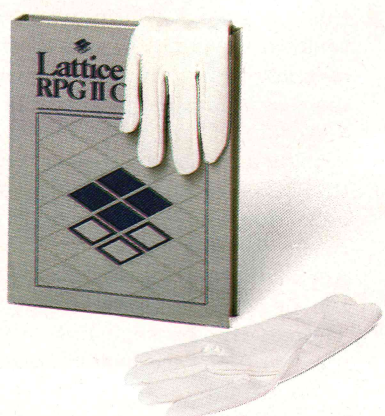
Lattice Bulletin Board Service
LBBS is our 24-hour a day bulletin board system that allows you to obtain notification of new releases, general information on Lattice products, and programs for the serious user. And if you've ever experienced the frustration of having to wait a year or more for a new release (that has corrected a bug), you'll really appreciate LBBS. Because with this service, you can actually download the latest program fixes to instantly eliminate any bugs discovered after release.

Technical Support Hotline
Responsible, dependable and capable Support Representatives are only a phone call away. You will talk to a highly skilled expert who is trained to answer any questions you have relating to specific Lattice products. Remember, your complete satisfaction is our goal.

McGraw-Hill BIX™ Network
The Byte Information Exchange (BIX) Network is a dial-in conference system that connects you with a Special Interest Group of Lattice users. The nominal one-time registration fee allows you to BIX-mail your questions—via your modem—directly to Lattice. Or you can post your questions in the conference mode for Lattice or other users to answer. Once again, you have 24-hour access.

You Also Receive:
▪ Timely updates and exciting enhancements ▪ 30-day, money-back guarantee ▪ Lattice Works Newsletter ▪ Technical Bulletins ▪ Access to Lattice User Groups

Lattice has developed more than 50 different Microcomputer software tools that are used by programmers worldwide. We were there for every MS-DOS release. We're there now for OS/2. And we'll be there for the next generation of technical changes. But most of all, Lattice is there for you.



Available through dealers and distributors worldwide.

Lattice
Subsidiary of SAS Institute Inc.

Lattice, Incorporated
2500 S. Highland Avenue
Lombard, IL 60148
Phone: 800/533-3577
In Illinois: 312/916-1600

CIRCLE NO. 158 ON READER SERVICE CARD

(continued from page 85)

more to do with the simplistic structure of a dBASE III file than with the Lattice documentation. A chapter that described the organization of dBASE .dbf and .ndx files would have been very helpful. Data is stored in strange ways and an in-depth description of these methods would also be useful. (There's a description of dBASE .dbf and .ndx files in Jeff Walden's *File Formats for Popular PC Software* [New York: Wiley, 1986.])

Some of this information can be gleaned from the subroutine descriptions, but I'd like it all in one place. The manual is poorly organized and hard to use as a reference. For example, dBC functions naturally break into four categories: functions to manipulate the database itself, functions to manipulate the index files, functions to manipulate the memo files, and various formatting functions. The manual, however, is organized alphabetically (generally my preference), but there is no functional index. As a consequence it's hard to find the function you need if all you know is what it does. The subroutine naming conventions are so strange that it's hard to guess what the name will be. The problems with the documentation are compounded by occasional errors in the examples. (One of them has a couple of stars where none belong, an error that caused me about ten minutes of unnecessary head scratching.)

The subroutines themselves are sometimes rather low level. The low-level functions should be included in the package, but higher-level ones are really required, too. For example, there's a host of functions that convert strings and numbers to and from the formats required in dBASE files. Fine, but as a C programmer, I want *sprintf( )*- and *sscanf( )*-like subroutines that would take care of all the details of the conversion for me. Ideally, you'd pass these functions a database pointer, a format string that identified the fields you wanted to access, and several objects to convert—something like this:

```
dprintf( db_file,
         "%name %street %zip",
         "J. P. Morgan", "Easy St.", 09311 );
```

and the function would do the rest for you. No such function is provided, however.

The dBC programming interface is very amateurish and harder to use than is necessary. I'd expect this sort of thing from users' group software but not from professional programmers. There are several problems. First, Lattice has implemented functions with C-like names but with nonstandard interfaces. For example, the function used to open a database is obviously modeled after *open( )*. It's called with:

The package does the job and is easy enough to use.

```
int dBopen( filename, mode, dbffd )

char *filename;
int mode;
char **dbffd;
```

There are several problems here. The main one is that you get back the file descriptor by passing in a pointer to a place to put it (*dbffd*). *DBopen( )* returns an error code, but that's it. I also don't like the declaration of a file descriptor as character pointer. It obviously isn't a string and shouldn't be declared as such. I'd prefer either a *void* pointer or a special type (such as *FILE*).

Were I to rewrite the interface to this function, it would look like this:

```
DBFILE *dBopen( filename, mode )

char *filename;
int mode;
```

It would return –1 on error and a file descriptor on success; there

C CHEST
(continued from page 87)

would be a *db_errno* that would hold an error code and a *db_perror()* that would print an error message. That is, the dBC function should be identical to *open()*, except that it should open a database rather than a normal file. If you're going to use this package a lot, it would be worthwhile to write a glue library that mapped the weird calling conventions to something more reasonable.

The second problem is the function names themselves. The names are the too-cryptic abbreviations popular with inexperienced C programmers (such as *_dbcmsiz()* or *dBdbfbuf()*). There seems to be a rationale to the naming conventions, but that rationale is never made clear.

Finally, the interface has several built-in inefficiencies. For example, many values are returned from subroutines in a *char* rather than an *int*. (Usually you pass in a pointer to a *char*, which is filled by the subroutine.) This is a classic mistake made by novice C programmers who think they're saving space by shoving data into *char*s. They're wrong. The compiler must convert all *char*s to *int*s before they can be used in an expression. This conversion is always performed, even in expressions that use nothing but *char*-size variables. Not only does the type conversion take time but also the extra code needed for the conversion is usually far larger than the data space saved. The only valid use for C's *char* type is an array of the things.

Frankly, the poor-quality interface makes me wonder about the quality of the rest of the code. The package does work and has been reliable in all the applications that I've written, but I don't really trust it.

In spite of the foregoing, though, I recommend this product to those of you who need to write Ashton-Tate-compatible programs. I intend to keep using it myself in these situations. The package does the job and is easy enough to use. It lets you put together a working program in a reasonable amount of time with a minimum of fuss. Moreover, you can do it all in C, without having to

learn yet another programming language. To make dBC really usable, however, you'll need to add a layer of glue functions to make the program interface more reasonable.

I wouldn't use dBC III Plus if I didn't need dBASE compatibility, however. The ISAM model used by Ashton-Tate is not great for most applications. The databases themselves are poorly structured, generally hard to use, and slow. (The speed is Ashton-Tate's problem, not Lattice's. Any program made with Lattice's package will be an order of magnitude faster than the equivalent dBASE III program. Nonetheless, a database program that used a different model altogether would probably be faster still.) Moreover, the Lattice package is marred by the poor quality of both the documentation and the programming interface. Though you could use dBC for any general-purpose database program, I wouldn't.

Availability

All source code for the multitasking kernel described both this and last month (including the priority-queue stuff) is available for $30 on an IBM PC 5¼" disk from Software Engineering Consultants, P.O. Box 5679, Berkeley, CA 94705. Include local sales tax if you're ordering from California.

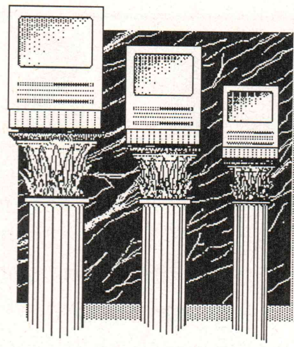
In addition to the kernel code and the priority-queue routines, the disk includes an enhanced version of the curses window I/O package described in the July 1987 C Chest. Because the enhanced curses uses direct video reads and writes rather than going through DOS, it's useful in multitasking applications that can't use the DOS I/O functions. This version of curses supports overlapping windows (though you can only write to the top one) and lets you delete and move windows. In addition, it lets you create boxed windows (Unix's curses doesn't).

The (unmodified) priority-queue routines are also available on CompuServe in DL1 of the DDJ FORUM. The file is called QUE2.C.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 3.**

Introduction, Mac Sources, Lightspeed C, and Code Corner



It's been four years now since the Macintosh popped into view, heralded by that great Ridley Scott Superbowl commercial. Though there was obvious brilliance to the design, there were also strong whiffs of arrogance and hype. But, hey, I've been accused of the latter myself. There was nothing to do but pop out of the hills and take a closer look.

I cruised over the Siskiyous to see my Apple dealer buddy, John Manzer. I got to the store, chewed the fat a few minutes, scanned the marketing propaganda, nosed the technical specs, then plunked down at the machine. Cynical musings twisted my mind, but what the hell, let's start 'er up.

I didn't like the locked hardware. I didn't like the one drive. I didn't like the lack of a hard disk. I didn't like the price. I didn't like the (hah hah) wide selection of printers. I didn't like the yuppistic overtones.

But I loved the machine. It was fun to use. Oh, there were flaws, but they seemed minor compared to what the creators got right. Above all, the interface snapped. It didn't have a speed snap—not yet—but rather the feel-of-a-fine-tool kind of snap. The Macintosh communicates via clean visual metaphors, and that's a channel with a lot of bandwidth. Something about the Macintosh interface just feels good, like soft light filtering through a redwood forest or playful kittens careening and bouncing about the world. People needed to use computers like this. I needed to program computers like this. Yow!

by Stan Krute

Four years of good nurturing has led to some lovely growth. We've got a wide array of languages, detailed system documentation, the LaserWriter, Mac IIs, Hypercard, MultiFinder, and some remarkable application software. The platform's been consolidated, and the best is yet to come. Happy birthday, Mac! Blow out those candles, eat up that cake, chug down the Jolt, get crazy with your buddies. Hell, Maddie Hayes's got you in her office: Y'all done good.

The Doc Gets a New Column

And so *Dr. Dobb's* gets a Mac column. What'll I do here?

1. Take note of well-done applications and extensions to the user interface.
2. Review a wide array of Mac programming tools: software and works on paper.
3. Talk with, and about the work of, innovative Mac programmers.
4. Discuss some of the more interesting algorithms and data structures contained in the Mac ROM/OS. This thing's a graduate course in programming, with interesting tidbits lurking between every *LINK/UNLK* pair.
5. Write some code. Mac programming's the most addictive fun I've had in the innards of a machine. The universe of the Mac ROM/OS is quite dynamic, so there's a premium—nay, an imperative—on programming that's clean, concise, and careful. Yield to that imperative, then combine it with an interface design that syncs with the Mac paradigm, and you get applications that not only work but that are also fun, easy to use, empower your users, and smack of elegance.
6. Provide access details. I'll always give you a box (see page 106, for example) filled with information that'll help you get hold of items mentioned in that month's column.

Reviews, Criticism, Objectivity

It's a lot of hard work to get a book or software product on the market. I feel a special obligation to creators to be scrupulously fair with any review/comments/criticism of a work. Print's powerful stuff. If I think something is seriously flawed, I won't even bother to mention it here; I prefer to send a quiet note detailing my qualms directly to the publisher. I'd rather put this column's energy into feeding awareness of the good stuff.

A note on objectivity: I'm lucky enough to know and/or have worked with some of the people whose products I may mention. But it does nobody any good if I let that shade my opinions. On the other hand, I don't want to ignore a good product just because I've had something to do with it. So I'll always mention any close connections I've got to a particular item in an objectivity note. Just know that it's done to help you weigh my opinions, not as name dropping.

Getting Up To Speed

The code samples I'll be showing aren't for raw beginners. This is *DDJ*, after all. But it's easier to get up to Mac programming speed now than it was in the early days. A lot of resources are available to help you cruise the learning curve. Here's a minimal list:

1. Join APDA, the Apple Programmer's and Developer's Association. Godchild of Dan Cochran and Dave Lingwood, this is a one-stop source for draft and finished copies of Apple documentation and development tools as well as a wide variety of third-party products. Dues are a reasonable $20 per year, it has an

TO THE MACS
(continued from page 90)

800 phone number, and you can charge to plastic.

2. If you're developing commercial products, try to become a certified Apple developer. Most important, this gives you access to Apple's electronic-mail technical support. Within the corporate constraints, the remarkable tech support humans will help you work through most any problem. Answers come within 24 hours. Other certified developer pluses: marketing assistance, developers' conferences, discounts on development hardware, and a tinge of credibility.

3. Get *Inside Macintosh* and its descendants. If the Pulitzers had a technical writing category, *Inside Mac* would own a prize. Caroline Rose and her cohorts and descendants have given us the most comprehensive insight into a complex cybernetic system yet seen. This is the starting point for all Macintosh programming. Take a look at the APDA newsletter for the latest volume count. The only flaw is a lack of practical examples, but other folks have filled that gap (see next item).

4. Add at least the following five books to your library: Scott Knaster's *How to Write Macintosh Software*, Dan Weston's *The Complete Book of Macintosh Assembly Language Programming (Volumes I and II)*, *The Best of MacTutor*, and *The Complete MacTutor*. Other fine Mac programming books are available, but these five are classics. They give you the practical examples that *Inside Macintosh* lacks. And, if you share my lack of photographic memory, you'll also want some language references. I like the handy little *Signetics S68000 User's Guide* for 68000 assembly language and Harbison and Steele's *C: A Reference Manual*. (Objectivity note: Dan Weston is a longtime friend and fellow traveler.)

5. After you've reupped with *DDJ*, subscribe to *MacTutor*. It's one great Macintosh programming magazine, filled each month with nerdly little programming goodies.

6. Put together an array of development tools. Plenty of good ones are available, and every now and then, I'll review some here in the column.

After all, I'm a language junkie; one of the perks of this gig is feeding the addiction guilt-free.

Choosing development tools is pretty personal. With one exception (now justly dead in the market), I haven't hit a Mac programming tool that someone wouldn't find useful in some context. The Mac environment must have some kind of inspirational effect. You'll have to follow the usual path to find personally amenable tools: talk to friends, read reviews, scan the ads, ask questions on the networks, play around. For what it's worth, here's a commented list of what I currently find myself using; note that most of my Mac work is done in C, with 68000 assembly language for speed tweaks and writing specialized code resources.

- C compiler: Lightspeed C 2.11—blazingly fast, holds to standards, feels good.
- 68000 assembler: MDS 2.1—I started here and have found no reason to move on; now marketed as the Consulair 68000 Development System.
- Debugger: TMON 2.8—clean, simple, powerful, can survive a lot of weirdness.
- Text editor: QUED/M 2.04—solid, feature-packed, useful macro language, very nice.
- Resource editor: ResEdit 1.1B1—one of the unsung great hacks, this Apple-produced program is the Mac-like way to create and manage resources.
- Disk and file editor: Fedit Plus—does anything you can think of to disks and files, fast and accurate with a very clean interface. (Objectivity note: I worked on the latest Fedit Plus documentation.)
- Code snooper: MacNosy—allows intelligent examination of any piece of code you can specify, including and especially the ROM. (Objectivity note: I worked on the [little yellow book] MacNosy documentation.)

7. Get a Mac with a hard drive and as much memory as you can afford. Anything less will drive you nutso fast. Hey, I oughtta know: I did my first Mac programming in assembly language on a 128K one-drive machine with 8-minute turnarounds. With a language such as Lightspeed C or Turbo Pascal on a multimeg SCSI machine, turnarounds drop down into the sub-30-second range.

Useful Mail-Order Sources

When I'm not traipsing around civilization as a cybernetic nomad, I live in the middle of nowhere, so I have to rely on mail-order sources to get programming books, software, and miscellaneous supplies. I've found a couple of good ones I'm happy to share with you.

For books, I use Computer Literacy. This bookstore carries just about everything, takes credit cards, and ships UPS the day you order. For software and supplies, I use Computerware. It specializes in the Mac, also takes credit cards and ships UPS quickly, and has an 800 phone number. Both these places have retail outlets well worth a visit if you're in Silicon Valley.

Don't Trash Your Old Mac

Apple's Macintosh upgrade path has been a little bumpy. A lot of folks still have 512s, possibly upgraded from 128s, and wonder whether it's worthwhile doing any further upgrading. Here's what I did: got Apple's 800K drive/128K ROM upgrade ($300 at an Apple dealer), then added SuperMac's Enhance board

TO THE MACS
(continued from page 93)

($500 installed at Fry's Electronics). Enhance brings your Mac up to 2 megabytes (expandable to 6.5 with high-capacity SIMMs), gives you a slight speed increase, and adds a SCSI port and a small internal fan.

I had one problem with the parasitic clip that plugs Enhance into the motherboard's 68000, but SuperMac kept Federal Expressing me replacements until we had the problem licked—no problems since then. I get a big grin on my face when my original 128K Mac comes up on a speedy hard disk with megabytes of RAM at its disposal.

Lightspeed C

It's hard to hold in my feelings on this product. Let's just say this: I love it. Michael Kahl and the rest of the Think Technologies crew have given us something wonderful. This thing is fast—makes me want to stick some flame decals on the Mac. Working in the LSC environment's a tasty treat, and I for one refuse to go back to anything slower or less capable.

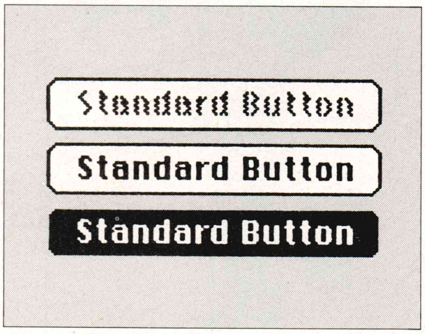


Figure 1: *Standard button in its three highlighted states—inactive, active, and highlighted*

Producing robust Mac code is (for me, at least) a highly iterative process. It bears repeating: Mac software exists in a very dynamic universe. The slightest coding miscue quickly propagates into screen-twisting madness. Debugging is tricky—well worth avoiding—so I like to write my code in snippets, testing and debugging each piece thoroughly before moving ahead. Lightspeed C, with its blazing turnaround speed, lets me do this painlessly.

A lot of attention's been paid to the product's details. Work goes on in a well-integrated project environment. Nitty little maintenance details are automated. The language, libraries, and header files hold closely to the relevant standards (Unix, Kernighan & Ritchie, Harbison & Steele, the evolving ANSI C, and *Inside Macintosh*). The editor's good—not quite so feature-laden as QUED/M but good—with powerful grep capabilities. The compiler puts out code that's fast and compact. It's easy to produce the various sorts of Macintosh code, with global and static variables available in each: double-clickable applications, desk accessories, device drivers, and code resources. In-line assembly-language code's allowed, with full access to the C name spaces. Resource file management is completely automated. Register variables are maximal: five data registers and three address registers. HFS and MultiFinder are well supported.

variation	content	border	highlighting via
0	text	outlined	inversion
1	text	outlined	content change
2	text	shadowed	inversion
3	text	shadowed	content change
4	PICT	bare	inversion
5	PICT	bare	content change
6	PICT	outlined	inversion
7	PICT	outlined	content change
8	PICT	shadowed	inversion
9	PICT	shadowed	content change
10	ICON	bare	inversion
11	ICON	bare	content change
12	ICON	outlined	inversion
13	ICON	outlined	content change
14	ICON	shadowed	inversion
15	ICON	shadowed	content change

Figure 2: RectCDEF's *16 button variations*

I recently had the pleasure of spending a September afternoon at Think headquarters doing free-form nerd talk with Michael Kahl (the prime Lightspeed C programmer), Andrew Singer (head of Think and coconceiver of Lightspeed), and Doreen Duplin (marketing/communications whiz). These are nice people in whom the joy of the great hack runs deep. Interesting backgrounds: Michael was a philosophy grad student before succumbing to the lure of machine logic. Andrew's known to many of us for his classic (and, sadly, out of print) Sherlock Holmes pastiche programming books *Elementary BASIC* and *Elementary Pascal*.

Recent releases of both Lightspeed C and Lightspeed Pascal (2.13 and 1.11A, respectively, as this column is written) have been maintenance releases, keeping the languages current with the latest Mac machinery and system software. In the works, though, are major new releases of both languages. Look for greater speed and, for Lightspeed C, powerful debugging capabilities. "It's time for another dose of the spectacular," quoth the Singer.

I wish I had room to give you a complete transcript of the afternoon's conversations. The Thinkers said a lot of smart stuff. Maybe in a future column. Meanwhile, take this as a bottom line: if you program the Mac in C, check out Lightspeed.

Code Corner

All right, time to get down to a little code hacking. My first project involves writing and using a custom control definition. Because of space constraints, I'll describe the project in two phases, continuing the discussion in next month's column.

Macintosh applications are rife with controls: buttons, scroll bars, check boxes, radio buttons, et al.

TO THE MACS

(continued from page 94)

Definitions for standard controls are built into the Mac ROM/OS—there's the standard button, for example. Clicking a standard button with the mouse makes things happen. Standard buttons have three basic states, each with a corresponding visual metaphor: inactive, when the button won't respond to a mouse click; active, when the button will respond to a mouse click; and highlighted, when the button's in the midst of being clicked. Figure 1, page 94, shows a standard button in each of these three states.

But you also have the ability to define, via a CDEF code resource, your own buttons. The CDEF resource can then be incorporated into an application and can be called upon whenever the application wants to put a button on the screen. Custom CDEFs are not very difficult to write and can provide a lot of flexibility at low memory cost. The CDEF I'll be showing you in this column, for example, is less than 1,400 bytes long yet it provides 16 new types of buttons—that's less than 88 bytes per button variation. Such a deal.

Development Details

I wrote my CDEF, called *rectCDEF*, in assembly language using MDS 2.1. That's because I wanted high execution speed and small code size. I wrote a demo application in C that shows off the 16 button types using Lightspeed C 2.11. Resources for the application were put together with ResEdit 1.1B1. PICTures for particular buttons were drawn in SuperPaint 1.0p, then transferred into ResEdit via the Scrapbook.

I first got the demo application up and running, albeit with just one control, that being of button variation 0 (see later). Then I started work on the CDEF. As I worked on the CDEF, I used an Exec JOB file to assemble the code, link it, turn it into a resource, then merge that resource into the demo application for testing. I worked on one variation at a time, adding a control of that type to the demo application, then fixing the CDEF to cover that case.

Any particular CDEF can have up to 16 variations. (Actually, you can hack in a few thousand, but that technique's for another article.) I used all 16 in *rectCDEF*. The *rectCDEF* buttons live in a rectangular world. A particular button variation can contain text, a picture, or an icon. Text can be in any font/size/style combination the Mac's capable of. A button variation can have a simple outline, a shadowed outline, or (unless it's a text variation) no outline. A button variation can indicate highlighting via inversion or a change of content.

Figure 2, page 94, details the 16 *rectCDEF* button variations. Figure 3, page 98, shows examples of each variation, with pictures of the active and highlighted states.

An Overview

The demo program, imaginatively named custom controls demo, puts up a modal dialog containing examples of *rectCDEF* buttons, then responds to button clicks. Figure 4, page 100, is a screen snapshot of the program's modal dialog. Using a modal dialog simplified the program's event-handling logic; it's a nice technique for bench testing new routines.

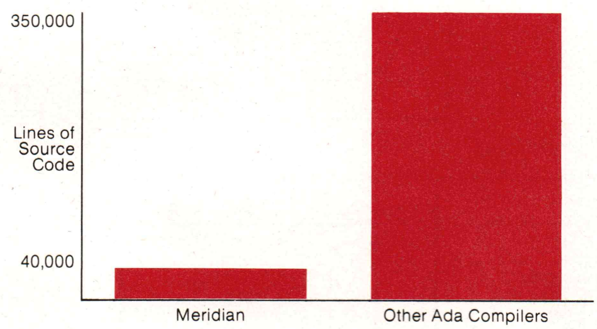
Figure 5, page 100, shows the files

THE ADA® WORLD HAS CHANGED.™

Next Generation Ada Technology

Third generation Ada is here today with Meridian AdaVantage, and it's validated by the Department of Defense and their suite of 2,700 tests. Ada's first generation was the proving ground technology of the early eighties. Then came the large validated compilers of the mid-eighties, which were often inefficient and machine specific. Now Meridian offers a compact efficient Ada technology that is highly portable from PC's to minis to mainframes.

COMPACT MERIDIAN Ada IMPLEMENTATION



The chart above depicts the dramatic difference between Meridian's implementation and other comercially available Ada implementations. Meridian's order of magnitude smaller code size results in an extremely fast compiler that will produce highly optimized code.

Meridian's Track Record

Since 1981, we've been building compilers the old-fashioned way, through hard work and experience gained from our development of successful portable compiler products. These products are installed at over 1,500 sites, including almost all major DoD contractors as well as commercial software developers.

VALIDATED Ada

THIS PRODUCT CONFORMS TO ANSI/MIL-STD-1815A AS DETERMINED BY THE AJPO UNDER ITS CURRENT TESTING PROCEDURES

Price/Performance Breakthrough

The Meridian AdaVantage v2.0 validated Ada compiler costs $795 and provides a production quality Ada development tool.

SIEVE BENCHMARK

	Meridian v2.0	Alsys v1.3
Compile and link time	27 sec	59 sec
Execution time	4.6 sec	4.9 sec
Execution size	27,344 bytes	42,129 bytes
Price	$795	$2,995

NOTE: All times measured on an IBM at 5170 (8MHz) and a 4MB RAM card required by the Alsys system. When running without the RAM card, the Meridian compile and link time is 46 seconds.

The Meridian AdaStarter incorporates all of the features of AdaVantage, with certain size limitations. AdaStarter is perfect for anyone who wants to learn how to program in Ada. The complete $99 cost is applicable toward the purchase price of the AdaVantage production compiler.

The compilers all run in a standard PC configuration with 640K of memory, a hard disk, and DOS v2.1 or higher.

To order today, or get more information, call 1-800-221-2522 (outside California) and 1-714-380-9800 (inside California).

"The Meridian compiler is a very well-thought-out compiler. The compiler is fast and execution speed is more than adequate... Overall, we'd rate the Meridian compiler as very solid."
— COMPUTER LANGUAGE, DECEMBER 1986

"The more affordable AdaVantage v1.0 is good for the average programmer because of its price, the extent of its implementation, and its relaxed hardware requirements... [AdaVantage v2.0] should be competitive with Alsys Ada, since it will be a full Ada at less than a third of the price of Alsys."
BYTE, JULY 1987

MERIDIAN
SOFTWARE SYSTEMS, INC.

23141 Verdugo Drive, Suite 105, Laguna Hills, CA 92653
800/221-2522 (outside CA) 714/380-9800 (inside CA)
Telex: 650-268-0547 MCI Fax: 714/380-1683

CIRCLE NO. 166 ON READER SERVICE CARD

(continued from page 96)

involved in the demo program. Custom controls demo PROJ is an LSC project file that contains the C source code file; custom controls demo.c (see Listing One, page 54); and MacTraps, the LSC library that hooks code into the Mac ROM/OS (see Figure 6, page 100). Custom controls demo.h, in Listing Two, page 64, is a file of private definitions for custom controls demo.c. Custom controls demo PROJ.rsrc (available on CompuServe and the *DDJ* listings disk) is a collection of program resources, including *rectCDEF*, that gets bound into the final application. It was put together with ResEdit. Finally, custom controls demo

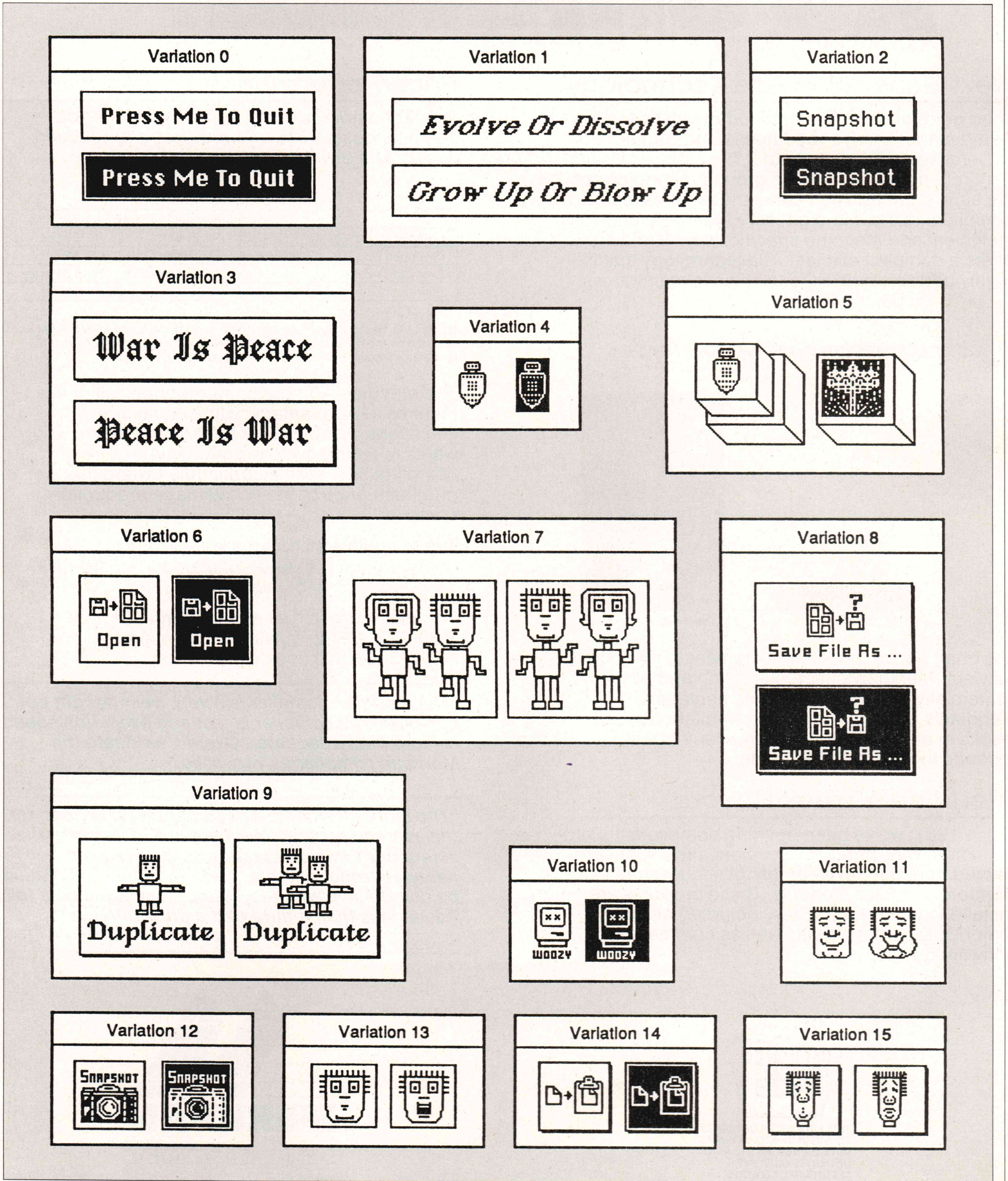
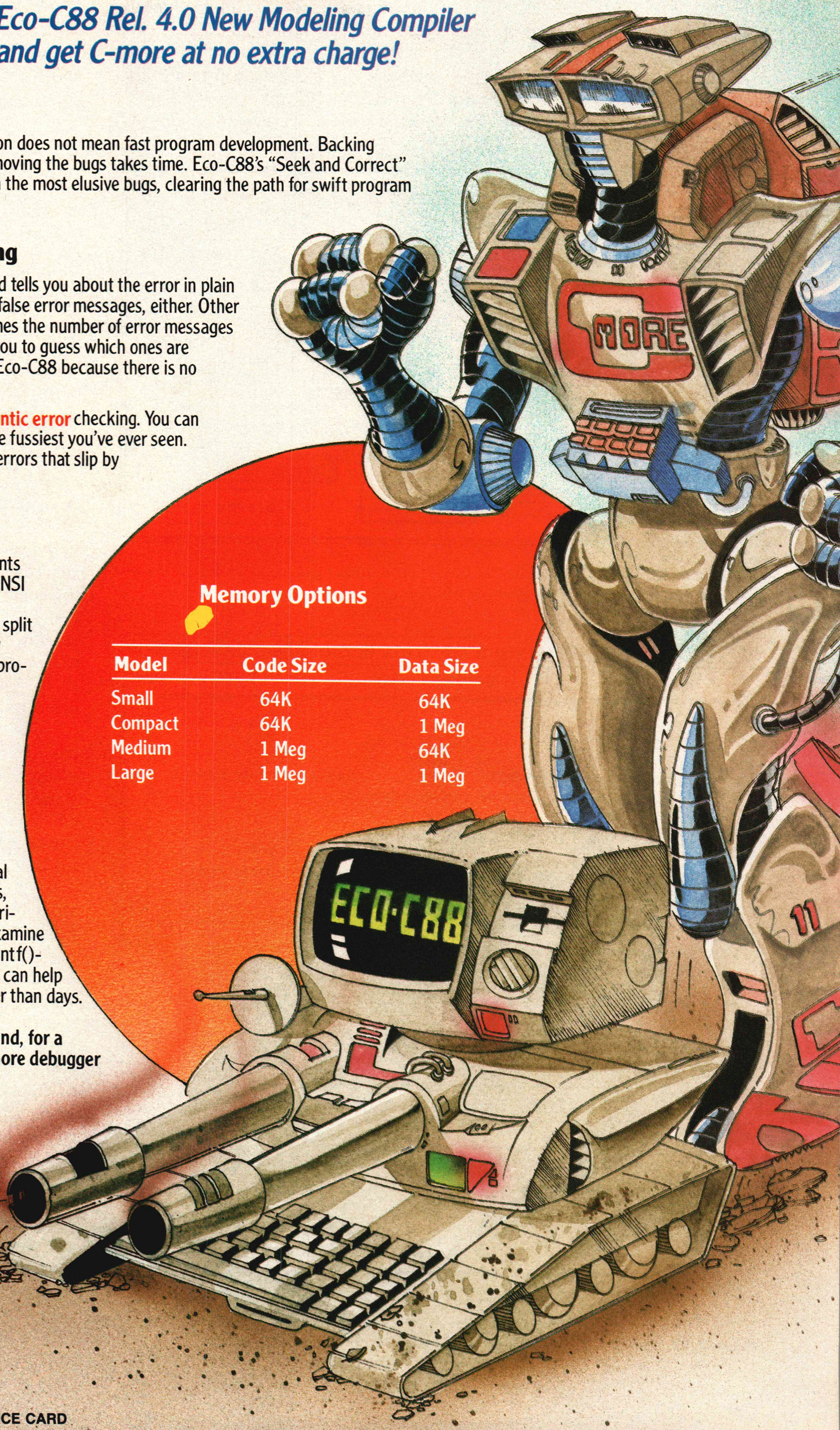


Figure 3: *Samples of the 16* rectCDEF *variations in active and highlighted states*

DEBUGGING SWAT TEAM

Order Eco-C88 Rel. 4.0 New Modeling Compiler and get C-more at no extra charge!



Seek and Correct

You already know that fast compilation does not mean fast program development. Backing up for bogus error messages and removing the bugs takes time. Eco-C88's "Seek and Correct" three - way error checking finds even the most elusive bugs, clearing the path for swift program development.

Double Barrel Error Checking

Eco-C88 nails **syntax errors** cold and tells you about the error in plain English. And there's no avalanche of false error messages, either. Other compilers can generate up to four times the number of error messages actually present; they leave it up to you to guess which ones are real. You'll be more productive with Eco-C88 because there is no guess work.

Eco-C88 provides ten levels of **semantic error** checking. You can select from almost no checking to the fussiest you've ever seen. Eco-C88's "picky flag" finds subtle errors that slip by other compilers.

Eco-C88 also features:

- All data types, plus ANSI Enhancements
- Robust library, including many new ANSI functions
- CED editor with online function help, split windows, compile-edit-link capability
- New, expanded manual with sample programs for the library functions

Memory Options

Model	Code Size	Data Size
Small	64K	64K
Compact	64K	1 Meg
Medium	1 Meg	64K
Large	1 Meg	1 Meg

C-more Source Code Debugger

Finally, if a really nasty bug persists, put C-more, our source code debugger, to work. With C-more you can watch your program as it executes, single-step it, set simple or conditional breakpoints, test complex expressions, use variables as indexes into other variables, initialize and trace variables, examine CPU registers, display results with `printf()`-type options and much more. C-more can help you track down bugs in minutes rather than days.

The price for Eco-C88 is $99.95. And, for a limited time, we'll give you our C-more debugger at no extra charge.

Ecosoft Inc.
6413 N. College Ave.
Indianapolis, IN 46220

(317) 255-6476 (Tech Info)
(800) 952-0472 (Orders)

CIRCLE NO. 167 ON READER SERVICE CARD

TO THE MACS
(continued from page 98)

is the double-clickable final application.

Although it's small and simple, custom controls demo.c follows the classic pattern of Macintosh programs. First come a few setup activities. Then the program sits in a main event loop, waiting for events of interest. When such an event occurs, the program figures out what's up, acts appropriately, then returns to the main event loop. At some point an event occurs that tells the program to pop out of the main event loop. Then come a few cleanup activities, followed by an exit to the OS shell.

Each control button in custom control demo's main modal dialog has a corresponding item number in the DITL resource that supplies the dialog. Figure 7, page 101, matches each item with its DITL item number. These numbers are given symbolic names in the header file custom control demo.h. The same numbers are used for the CNTL template resources that each DITL item points to.

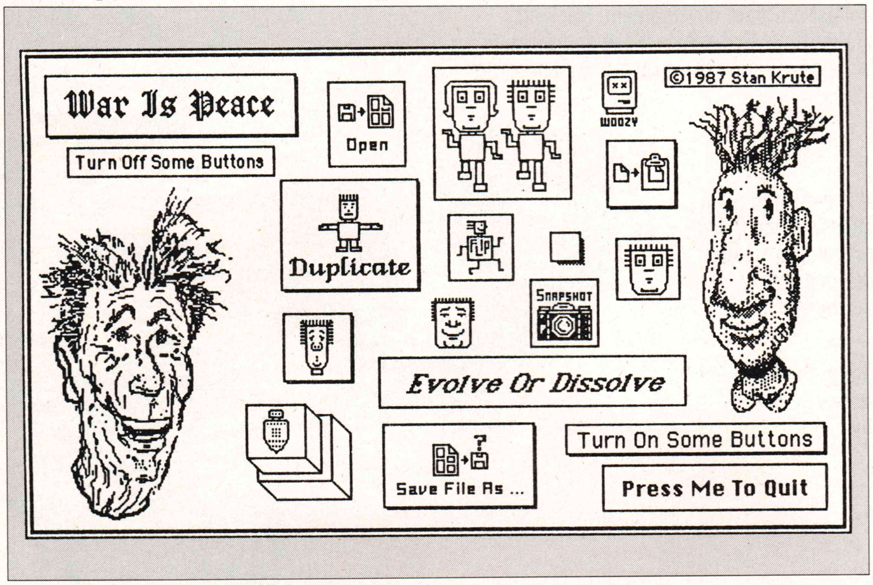


Figure 4: *Custom control demo's main modal dialog*

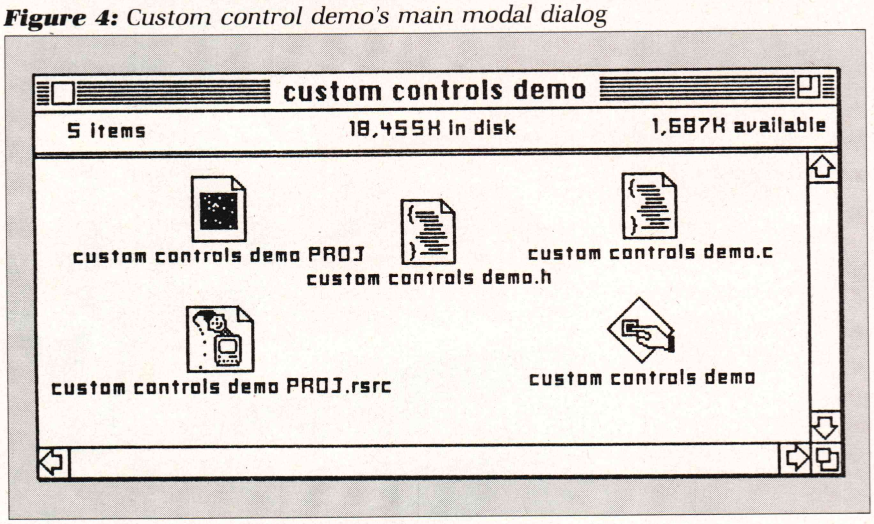


Figure 5: *Five files used by Lightspeed C to build the program custom controls demo*

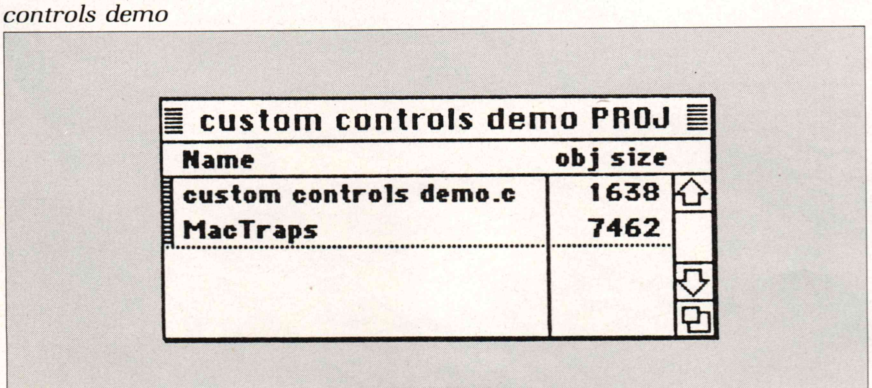


Figure 6: *The LSC project file used to build custom controls*

A Few Function Notes

If you've done your homework, the demo program should seem trivially simple, so I won't go into massive descriptive detail. That'll get saved for next month when it's time to cruise the *rectCDEF* assembly-language code. Here are a few notes:

main—Sets up the Mac managers, gets the modal dialog going, runs the main event loop, then cleans up and exits when all is done. Note the substitution of the ROM call *ModalDialog* for the usual *GetNextEvent* as the heart of the program's main event loop.

inititializeManagers—Grabs some master pointers, forces the heap to grow and clean itself, gets the ROM/OS managers up and running, flushes the event queue, and brings up the standard arrow cursor.

studyAndSetEnvironment—Figures out the size of the screen and menu bar. This information is used later on to position windows neatly.

getThatDialogCookin—Brings the main modal dialog into memory, positions it on the screen, sets its font to Geneva 12, then makes it visible. Note well: it's a good idea to use dialog and window templates that come up invisibly. Then you can bring them into memory, pull off any adjustments in private, and use *ShowWindow* to make them appear.

dealWithDialogItem—Just a big switch statement to case out on the button that got clicked in the main event loop. The top layers of a Mac

application are usually filled with such switch statements as the program zeroes in on exactly what kind of event occurred and what to do about it. Note how the *quitItem* button controls the main event loop via the global Boolean variable *finished.*

The following nine routines deal with the clicks of specific buttons:

doOrwellItem—The *orwellItem* button stays highlighted while the *ronItem* button fades in and out.

doSnapshotItem—The *doSnapshotItem* button lets you take action pictures of the demo program via a call to a Camera desk accessory. If you don't have such a DA in your system file, the *OpenDeskAcc* call returns without crashing.

doMushroomItem—Similar to *doOrwellItem*. This time the *bumperStickersItem* fades in and out of view.

doOpenItem—Calls on the standard file-opening routine, then does nothing with the routine's result.

doSaveAsItem—Calls on the standard file-saving routine, then does nothing with the routine's result.

doFlipItem—Takes a list of content-changing buttons, then runs them through a little animation routine by turning highlighting on and off.

doSomeOffItem—Takes a list of buttons and makes them inactive.

doSomeOnItem—Takes the same list of buttons passed to *doSomeOffItem*

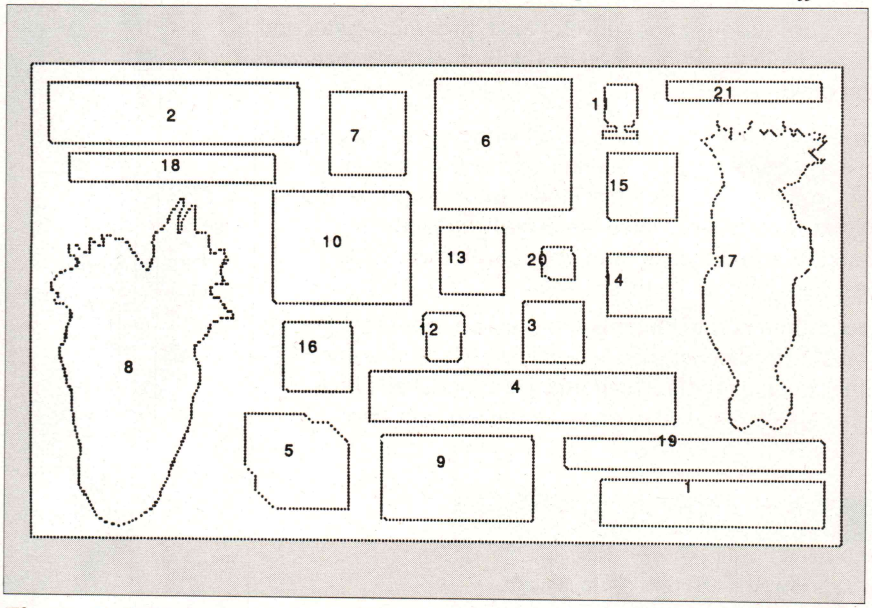


Figure 7: *DITL item numbers for each of the controls in custom control demo's main modal dialog*

PROGRAMMER'S PARADISE PRESENTS

C Tools

by Phoenix

Power Tools for Serious Programmers

If you are serious about developing C programs faster and more efficiently, Phoenix offers the high performance tools you need.

Professional programmers worldwide rely on Phoenix tools to more quickly write, test and deliver the tightest, cleanest C applications possible. Power tools. Tools like Plink® 86*plus* and PforCe,™ which are established industry standards, and Pfinish™ and Pre-C,™ which are today's best kept programming secrets.

Programmers rate Phoenix tools as superior for many reasons. Performance. Ease of use. Flexibility. Integration with other Phoenix tools. The bottom line is that all Phoenix tools are engineered to enhance the way you work.

PforCe™

- Features two fully detailed manuals.
- Contains over 400 pre-coded, optimized, object-oriented C routines.
- Includes databases with B-trees, windows, interrupt-driven communications, string handling, menus, all of the basic DOS interfaces, and a complete set of low-level functions to interface directly to the hardware.
- Comes complete with indexed reference manual, on line resident help, and quick reference card.
- Supports all memory models of the following C compilers: Lattice, Aztec, Microsoft, CI-86, and Wizard.
- Includes full source code.
- No royalties on generated applications using the libraries.

List: $395 Ours: $209

Pre-C™

- Now twice as fast!
- Cross-checks multiple source files and libraries at once.
- Reports incorrect, obsolete, and non-portable C usages.
- Uncovers errors in interfaces between program modules.
- Accepts full UNIX System III C syntax as well as ANSI proposed extensions.
- Uses external libraries with or without source code.
- Includes libraries for the latest releases of the Mark Williams, Lattice, CI-86, Microsoft, Wizard, and Aztec "C" compilers. Supports all memory models.

List: $295 Ours: $155

C/Pac

- Special Pre-C and PforCe combo pack

List: $495 Ours: CALL

Pfinish™

- Helps "fine-tune" programs by identifying inefficient sections of code that need to be written for maximum performance.
- Analyzes programs during execution.
- Produces reports and histograms that give a snapshot of which routines were reached, their callers, number of executions, amount of time spent in each, number of instructions executed in each, and more.
- Writes histograms and tabular reports, sorted by address or symbol, in any page width or height, to a file, the console, or the printer. Uses symbol table information to produce meaningful analyses on overlays and interrupts.

List: $395 Ours: $209

Plink® 86*plus*

- The only linkage editor containing advanced overlay capabilities.
- Handles any compiler or assembler producing standard Intel or Microsoft OBJ files.
- Ensures sample capacity for symbol and common block names (nearly 33,000).
- Supports an unlimited size file, an unlimited number of modules and up to 4,095 overlays nested up to 32 deep.
- Merges object modules, caches overlays in extended or expanded memory, and automatically reloads overlays upon function return.
- Includes Plib86 object library manager.

List: $495 Ours: $275

PforCe™++

- The only C++ library available on the IBM PC.
- Provides everything necessary to build complete applications.
- Contains high-level classes like windows, databases, B-trees, fields, menus, rings, lists, communication tasks, time/date stamps.
- Includes both high-level object classes and low-level hardware, BIOS and DOS access; complete source code to libraries with no royalty; disk management, archiving, compilation, and library management utilities supplied in both executable and source form; and overridable functions with reasonable defaults.

List: $395 Ours: $209

Find out why so many programmers rely on Phoenix for high performance,

1-800-445-7899
In NY: 914-332-4548

Programmer's Paradise™

CIRCLE NO. 169 ON READER SERVICE CARD

Call or write for the latest catalog

DISCOVER PARADISE

Programmer's Paradise Gives You Superb Selection, Personal Service and Unbeatable Prices!

Welcome to Paradise. The microcomputer software source that caters to your programming needs.

Discover the Many Advantages of Paradise . . .

- Lowest price guaranteed
- Latest versions
- Huge inventory, immediate shipment
- Knowledgeable sales staff
- Special orders
- 30-day money-back guarantee

Over 500 brand-name products in stock — if you don't see it, call!

We'll Match Any Nationally Advertised Price.

		LIST	OURS
386 SOFTWARE			
ADVANTAGE 386 C		895	799
ADVANTAGE 386 PASCAL		895	799
MICROPORT SYSTEM V/386 (COMPLETE)	*SPECIAL*	799	679
MICROSOFT WINDOWS/386		195	125
PHARLAP 386 ASM/LINK		495	419
PHARLAP 386 DEBUG		195	155
SCO XENIX SYS V 386 (COMPLETE)		1495	1195
VM/386	*SPECIAL*	195	119
X-AM		595	535
ARTIFICIAL INTELLIGENCE			
ARITY STANDARD PROLOG		95	79
MICROSOFT LISP		250	155
PC SCHEME		95	85
SMALLTALK/V	*NEW* V. 2.0	99	79
TURBO PROLOG		100	65
TURBO PROLOG TOOLBOX		100	65
ASSEMBLERS/LINKERS			
ADVANTAGE DISASM.	*SPECIAL*	295	249
ADVANTAGE LINK		395	359
ASMLIB		149	125
EZ_ASM		70	65
MS MASM	*REBATE*	150	95
PASM86		195	109
PLINK86PLUS		495	275
RELMS CROSS ASSEMBLERS		CALL	CALL
UNIWARE CROSS ASSEMBLERS		CALL	CALL
VISIBLE COMPUTER 80286		100	89
BASIC			
DB/LIB		99	89
FLASH-UP		89	79
MACH 2		75	59
MS QUICKBASIC	*REBATE*	99	65
QUICKPAK		69	59
TRUE BASIC		100	69
TURBO BASIC		100	65
DATABASE TOOLBOX		100	65
EDITOR TOOLBOX		100	65
TELECOM TOOLBOX		100	65

XENIX/UNIX PRODUCTS

MICROPORT & SCO PRODUCTS	CALL	CALL
ADVANTAGE C++	695	CALL
BTRIEVE	595	455
C-TERP	498	379
INFORMIX ESQL/C	749	CALL
INFORMIX 4GL	1500	CALL
INFORMIX SQL	995	CALL
KORN SHELL	125	115
MICROSOFT LANGUAGES	CALL	CALL
PANEL	625	535
PANEL PLUS	795	675
REAL TOOLS	149	89
RM/COBOL	1250	949
RM/FORTRAN	750	549
SCO MULTIVIEW (286)	395	319
SCO MULTIVIEW (386)	495	399

C++		
ADVANTAGE C++	495	479
PFORCE++	395	209
C COMPILERS		
C86PLUS	497	375
HIGH C	595	CALL
LATTICE C	500	265

		LIST	OURS
MICROSOFT C		450	269
QUICK C	*CALL-REBATE OFFER*	99	65
TURBO C		100	65
C INTERPRETERS			
C-TERP		298	219
INSTANT C		495	369
RUN/C		120	79
RUN/C PROFESSIONAL		250	155
C LIBRARIES			
BASIC_C		175	129
C ASYNCH MANAGER		175	135
C-FOOD SMORGASBORD		150	95
W/SOURCE CODE		300	179
C/PAC (PFORCE, PRE-C)	*SPECIAL*	495	CALL
C TOOLS PLUS/5.0		129	99
C UTILITY LIBRARY		185	119
C-XPERT		295	249
ESSENTIAL COMMUNICATIONS		185	119
COMMUNICATIONS PLUS		250	189
GREENLEAF C SAMPLER	*SPECIAL*	95	69
GREENLEAF COMM LIBRARY		185	125
GREENLEAF FUNCTIONS		185	125
MULTI-C		149	135
PFORCE		295	209
RESIDENT C W/SOURCE		198	149
TIMESLICER		295	265
W/SOURCE CODE		1000	895
TURBO C TOOLS		129	99
COBOL			
E-Z PAGE	*NEW*	295	259
MICRO FOCUS PRODUCTS		CALL	CALL
MICROSOFT COBOL		700	439
MICROSOFT SORT		195	129
OPT-TECH SORT		149	99
REALCICS		995	785
REALIA COBOL		995	785
W/REALMENU		1145	899
RM/COBOL		950	759
RM/COBOL-85		1250	999
RM/SCREENS		395	315
SCREENIO		400	379
DEBUGGERS			
ADVANCED TRACE-86		175	115
C-SPRITE		175	119
PERISCOPE I		345	279
PERISCOPE II		175	139
PERISCOPE III 8 MHZ		995	799
PERISCOPE III 10 MHZ		1095	899
PFIX 86 PLUS		395	209
T-DEBUG PLUS		60	49
XVIEW86		60	49
DISK/DOS/KEYBOARD UTILITIES			
BACK-IT		100	89
BOOKMARK		100	79
COMMAND PLUS		80	69
DISK OPTIMIZER		60	55
INTELLIGENT BACKUP		150	135
NORTON COMMANDER		75	55
ADVANCED NORTON UTILITIES		150	99
PDISK		145	99
FILE MANAGEMENT			
BTRIEVE		245	185
XTRIEVE		245	185
REPORT OPTION		145	99
BTRIEVE/N		595	455
XTRIEVE/N		595	455

		LIST	OURS
REPORT OPTION/N		345	269
C-TREE		395	315
R-TREE		295	239
C-TREE/R-TREE BUNDLE		650	519
DBC III		250	169
DBC III PLUS		750	595
INFORMIX PRODUCTS		CALL	CALL
PHACT MANAGER		249	219
XQL	*SPECIAL, NEW*	795	599
EDITORS			
BRIEF		195	CALL
W/DBRIEF		275	CALL
CVUE W/SOURCE CODE		250	199
EDIX		195	155
EMACS		295	265
EPSILON		195	149
KEDIT		125	99
LMK		195	139
PC/EDT		250	229
PC/VI		149	109
Pi EDITOR	*SPECIAL, NEW*	195	149
PMATE		195	109
SPF/PC		195	145
VEDIT PLUS		185	129
XTC		99	79
FORTRAN COMPILERS			
LAHEY FORTRAN		477	CALL
LAHEY PERSONAL FORTRAN 77		95	89
MICROSOFT FORTRAN		450	269
RM/FORTRAN		595	479
FORTRAN UTILITIES/LIBRARIES			
DIAGRAM'ER/DOCUMENT'ER		129	115
DIFF-E-Q		495	445
FORTLIB		125	109
FORTRAN ADDENDA		165	139
GRAFLIB		175	169
GRAFMATIC/PLOTMATIC		135	119
MATHPAC		495	445
NO LIMIT		129	115
SSP/PC		350	269
GRAPHICS			
ADVANTAGE GRAPHICS (C)		250	225
ESSENTIAL GRAPHICS		250	185
GSS GRAPHIC DEV. TOOLKIT		495	375
HALO		300	205
HALO (5 MICROSOFT LANG.)		595	389
METAWINDOW PLUS		275	229
TURBOWINDOW/C		95	79
TURBO HALO (FOR TURBO C)		100	79
LINT			
PC-LINT		139	99
PRE-C		295	155
MODULA-2			
FARBWARE MODULA-2		90	79
LOGITECH MODULA-2			
COMPILER PACK		99	79
DEVELOPMENT SYSTEM		249	CALL
TOOLKIT		169	139
WINDOW PACKAGE		49	39
ROM PACKAGE AND CROSS RUNTIME DEBUGGER		299	239
REPERTOIRE		89	75
TURBO PASCAL ADD-ONS			
ALICE		95	69
DOS/BIOS & MOUSE TOOLS		75	69
FLASH-UP		89	79

		LIST	OURS
METRABYTE DATA ACQ. TOOLS		100	89
SCREEN SCULPTOR		125	95
SYSTEM BUILDER		150	129
IMPEX		100	89
REPORT BUILDER		130	115
T-DEBUG PLUS		60	49
TURBO.ASM		99	69
TURBO ASYNCH PLUS		129	99
TURBO EXTENDER		85	65
TURBO HALO		129	99
TURBO MAGIC		99	89
TURBO OPTIMIZER		75	65
TURBO POWER TOOLS PLUS		129	99
TURBO POWER UTILITIES		95	79
TURBO PROFESSIONAL 4.0	*NEW*	99	79
TURBO WINDOW/PASCAL		95	79

FEATURED PRODUCTS

E-Z PAGE — Page formatter and forms generator for COBOL developers. E-Z Page automatically formats pages for the printer from within your COBOL program, and makes customizing forms easy with powerful page painting software. Supports all major PC COBOL compilers.
List: $295 **Special Price: $259**

Turbo PROFESSIONAL 4.0 — New library of over 400 routines for the latest version of Borland's Turbo Pascal (4.0). Includes pop-up resident routines, BCD arithmetic, virtual windows and menus, EMS and extended memory access, long strings, large arrays, macros, and runtime error recovery. Complete source code is included.
List: $99 **Special Price: $79**

XQL — Softcraft's newest product is an extension to Btrieve that provides relational database functions for your application. XQL features include data manipulation, definition, and data control languages, access to the system catalog, file security stored in the system catalog, automatic query optimization, end-user queries through simple menus, access from C, BASIC, and Pascal, commit and rollback commands for transaction recovery, and full LAN support for major networks.
List: $795 **Special Price: $599**

SMALLTALK/V NEW V. 2.0 — New version is a high-performance, production quality object-oriented programming environment. Includes:
*Advanced user interface featuring windows, pop-up menus and optional mouse.
*A set of tools for organizing and browsing the Smalltalk source code.
*An incremental program development capability.
*Bitmap graphics with optional color support.
List: $99 **Special Price: $79**

OPERATING SYSTEMS			
MICROPORT SYSTEM V/AT (COMPLETE)		549	475
SCO XENIX SYSTEM V (COMPLETE)		1295	995
WENDIN-DOS		99	79
OTHER MICROPORT, SCO, WENDIN PRODUCTS		CALL	CALL
PASCAL COMPILERS			
MARSHAL PASCAL		189	155
MICROSOFT PASCAL		300	185
PASCAL-2		350	329
TURBO PASCAL	*NEW* V. 4.0	100	65
TURBO PASCAL DEV. LIB.	*NEW*	395	259
BORLAND ADD-ONS		CALL	CALL
SCREEN DISPLAY/WINDOWS			
C-SCAPE		279	265
CURSES W/SOURCE CODE		250	169
GREENLEAF DATA WINDOWS		225	155
W/SOURCE CODE		395	269
HI-SCREEN XL		149	119
JYACC FORMAKER		495	449
JYACC JAM		750	679
MICROSOFT WINDOWS		99	65
MS WINDOWS DEVELOPMENT KIT		500	309
PANEL PLUS		495	395
PANEL FOR QUICKC / TURBO C		129	95
QUICKSCREEN		195	175
SCREEN ACE		195	159
SCREENSTAR W/SOURCE		198	155
VITAMIN C		225	159
VC SCREEN		99	79
VIEW MANAGER		275	199
WINDOWS FOR DATA		295	235
ZVIEW		245	169

HARDWARE PRODUCTS

AMDEK 722 MONITOR	750	499
AMDEK 730 MONITOR	899	569
AST ADVANTAGE PREMIUM W/512K	495	319
AST RAMPAGE! 286 W/512K	545	349
HERCULES GRAPHICS CARD PLUS	299	195
HERCULES IN COLOR CARD	499	329
IRMA 2	1195	779
ORCHID TURBO EGA	749	495
ORCHID TURBO PGA	1495	1099
VEGA DELUXE	379	259

ADDITIONAL PRODUCTS			
ADVANTAGE VCMS		379	329
BASTOC		495	399
CARBON COPY PLUS	*NEW*	195	139
DAN BRICKLIN'S DEMO PROGRAM		75	59
DB2C		299	CALL
FLOW CHARTING II		229	205
MAGIC PC		195	179
MKS TOOLKIT		139	115
NORTON GUIDES		100	65
PFINISH		395	209
POLYMAKE		149	125
POLYTRON PVCS		CALL	CALL
SOURCE PRINT		95	75
TREE DIAGRAMER		77	69

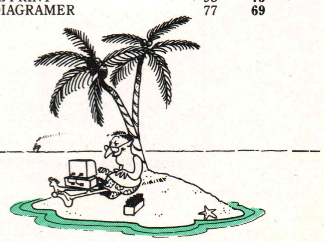
Terms and Policies
- **We honor MC, VISA, AMERICAN EXPRESS**
No surcharge on credit card or C.O.D. Prepayment by check. New York State residents add applicable sales tax. Shipping and handling $3.00 per item, sent UPS ground. Rush service available, prevailing rates.
- Programmer's Paradise will match any current nationally advertised price for the products listed in this ad.
- Prices and Policies subject to change without notice.
- Hours 9AM EST — 7PM EST

*Ask for details. Some manufacturers will not allow returns once disk seals are broken.

Corporate Buyers — Call for special discounts and benefits!

1-800-445-7899
In NY: 914-332-4548
Customer Service: 914-332-0869
International Orders: 914-332-4548
Telex: 510-601-7602

Programmer's Paradise™
A Division of Hudson Technologies, Inc.
42 River Street, Tarrytown, NY 10591



CIRCLE NO. 170 ON READER SERVICE CARD

TO THE MACS
(continued from page 101)

and makes them active.

doCopyrightItem—Brings up a modal dialog that expresses the author's interest in legal protection for works of art.

figureCenteredRectTLC—I don't know about you, but I go nuts over programs that don't know how to position things on different-size screens. This little routine shows how simple it is to be tidy.

To be continued next month.

Wrap Up

Special thanks go to the following for thoughts and actions that made this month's column possible: Tom Atkinson of Orchard Computer, Cynthia Bruschi of ICOM Simulations, Dan Cochran of Apple, Doreen Duplin of Think Technologies, Bruce Hammond of Starpoint Software, Michael Kahl of Think Technologies, Jerzy Lewak of Paragon Concepts, John Mitchell of Apple, David Perlman of Action Graphics, Andrew Singer of Think Technologies, Nathan Slemmer of Interstate Computer Bank, Tyler Sperry of *DDJ*, Mike Swaine of *DDJ*, Levi Thomas, and Dan Weston of Nerdworks.

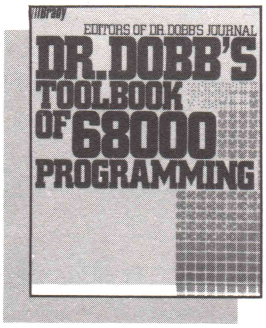
This is the first of my *DDJ* Mac columns. Feedback pro and con will be much appreciated; my access information is at the end of the column. Hot tips, keen insights, funny problems, and review copies of books and software are also solicited.

Next month for sure: assembly-language source for *rectCDEF* along with copious explanation and the rest of custom control demo's resources.

Next month maybe (depending on time, space, and circumstance): hypertalk secrets, living with multiFinder, macdraw with a brain, parasitic desk accessories, talks with various programming luminaries, and Microsoft madness revealed.

Stan Krute, when not serving as DDJ's new Mac columnist, is an artist, programmer, writer, and teacher. You can reach him via MCI

Programming Tips



Dr. Dobb's Toolbook of 68000 Programming

by the Editors of *Dr. Dobb's Journal of Software Tools*

This complete collection of practical programming tips and techniques for the 68000 family includes the best articles on 68000 programming ever published in *Dr. Dobb's Journal of Software Tools*, along with much new material. You'll learn about the most important features of the 68000 microprocessor from a full description of its history and design. Useful applications and examples will show you why computers using the 68000 family are easy to design, produce, and upgrade. Contents include:

● 68000 Instruction Set ● Bringing Up the 68000: A First Step ● A 68000 Cross-Assembler ● A Simple Multitasking Kernel for Real-Time Applications ● The Worm Memory Test ● A Mandelbrot Program for the Macintosh

All programs are also available on disk.

Book & Disk (MS-DOS, CP/M 8", Osborne, Macintosh, Amiga, Atari 520st)
Item #75-5 — $49.95
Book
Item #216649-6 — $29.95

Special Offer!

You SAVE 15%, and get a FREE MUG when you order the 68000 Programming Package, inlcuding ***Dr. Dobb's Toolbook of 68000 Programming***—with disk—and the ***68000 Cross Assembler***, all for only $63.95!

68000 Programming Package
Item #68pk — $63.95

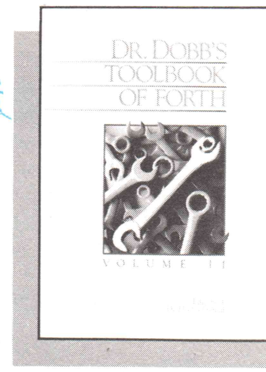
Specify Toolbook disk format (MS-DOS, CP/M 8", Osborne, Macintosh, Amiga, Atari 520st) and Cross Assembler format (MS-DOS, 8" SS/SD, Osborne)

68000 Cross Assembler

An executable version of the 68000 Cross-Assembler discussed in the book is also available, complete with source code and documentation. Requires CP/M 2.2 with 64K or MS-DOS with 128K.

Manual & Disk (8" SS/SD, Osborne, MS-DOS)
Item #71-2 — $25.00

SPECIAL OFFER



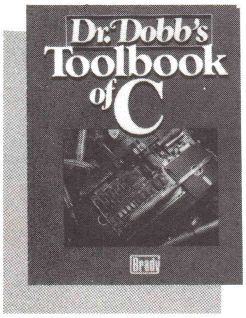
Dr. Dobb's Toolbook of Forth, Volume II

by the Editors of *Dr. Dobb's Journal of Software Tools*

This complete anthology of Forth programming techniques and developments picks up where the Toolbook of Forth, Volume I, left off. You'll find the best articles on Forth from *Dr. Dobb's Journal of Software Tools*, many of which have been updated and expanded. Respected experts in the Forth community also contribute examples of Forth used in many challenging and thought-provoking applications. Topics include: Forth philosophy and standards; system tools such as a Forth native-code cross-compiler for the MC68000 and a simple metacompiler; math tools, including Forth and the Fast Fourier Transform; Zen Floating Point; a Forth slide rule; sample applications such as Forth windows for the IBM PC, with code included; Programming Challenges, such as a Forth-Oriented, Real-Time Expert System, and much more. The screens in the book are available on disk as ASCII files.
Please specify MS-DOS, Macintosh, or CP/M (Osborne or 8" SS/SD).

Book & Disk (MS-DOS, Macintosh, or CP/M: Osborne, 8" SS/SD)
Item #51-8 — $45.95
Book
Item #41-0 — $29.95

C Tools

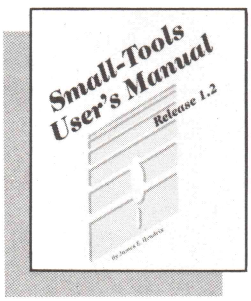


Dr. Dobb's Toolbook of C

by the Editors of *Dr. Dobb's Journal of Software Tools*

This authoritative reference contains over 700 pages of the best C articles and source code from *Dr. Dobb's Journal of Software Tools*, along with new material by C experts. You'll find hundreds of pages of useful C source code, including a complete compiler, an assembler, and text-processing utilities. Also included: all of Anthony Skjellum's C Programmer's Notebook columns distilled into one thought-provoking chapter.

Book
Item #615-3 **$29.95**

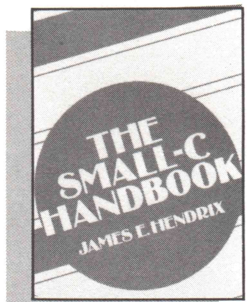


Small-Tools: Programs for Text Processing

by James E. Hendrix

This package of programs performs specific, modular operations on text files, including: editing; formatting; sorting; merging; listing; printing; searching; changing; transliterating; copying; concatenating; encrypting and decrypting; replacing spaces with tabs and tabs with spaces; counting characters, words, or lines; and selecting printer fonts. **Small-Tools** is supplied in source code form. With the **Small-C Compiler**, you can select and adapt these tools to your own purposes. Documentation is also included. Available for MS-DOS or CP/M systems. Please specify format.

Manual & Disk (MS-DOS or CP/M)
Item #78-X **$29.95**

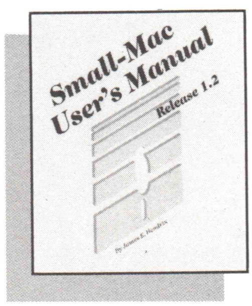


The Small-C Compiler and Small-C Handbook

by James E.Hendrix

This compiler and handbook provide everything you need for learning how compilers are constructed, and for learning C at its most fundamental level. You'll find a discussion of assembly language concepts and program translation tools, and learn how to generate a new version of the compiler. Full source code is included. The handbook and compiler on disk are available for both MS-DOS and CP/M systems. The handbook comes with an addendum for the MS-DOS version. Please specify format.

Book & Disk (MS-DOS)
Item #76-3 **$42.90**
Book & Disk (CP/M)
Item #67-4 **$37.90**

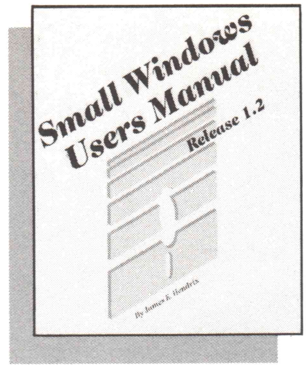


Small-Mac: An Assembler for Small-C

by James E. Hendrix

This assembler features simplicity, portability, adaptability, and educational value. The package includes: a simplified macro facility; C language expression operators; object file visibility; descriptive error messages; and an externally defined instruction table. You get the macro assembler, linkage editor, load-and-go loader, library manager, CPU configuration utility, and a utility to dump relocatable files. Documentation is included. For CP/M systems only. Please specify format.

Manual & Disk (CP/M)
Item #77-1 **$29.95**



Small-Windows: A Library of Windowing Functions for the C Language

by James E. Hendrix

Small-Windows is a complete windowing library for C (Microsoft 4.0 and Small-C). The package includes: 18 video functions written in assembly language; 7 menu functions that support both static and pop-up menus; 41 window functions to clean, frame, move, hide, show, scroll, push, and pop windows. Two test programs are provided as examples to show you how to use the library and the window, menu, and directory functions.
Documentation and full C source code is included.
Available for MS/PC-DOS systems for the following compilers: Microsoft C Version 4.0, Small-C, Lattice C and Turbo C.
Please specify compiler format.

Manual & Disk (MS-DOS)
Item #35-6 **$29.95**

SPECIAL PACKAGES

CP/M C Package

SAVE $27!

Receive: Dr. Dobb's Toolbook of C, The Small-C Handbook, the Small-C Compiler, Small-Mac Assembler, and Small-Tools Text Processing Programs. Only $99.95!

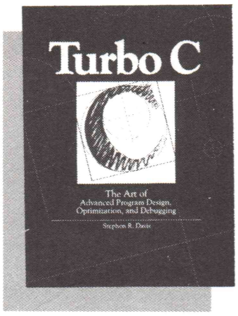
CP/M C Package
Item #005A **$99.95**

MS-DOS C Package

SAVE $22!

Receive: Dr. Dobb's Toolbook of C, The Small-C Handbook and MS/PC-DOS Addendum, the Small-C Compiler, Small-Windows, and Small-Tools Text Processing Programs. Only $109.95!

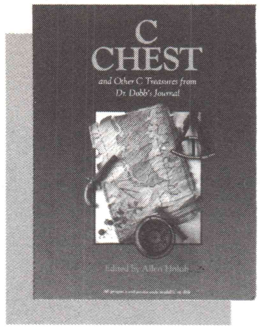
MS-DOS C Package
Item #005W **$109.95**



Turbo C: The Art of Advanced Program Design, Optimization and Debugging
by Stephen R. Davis

Overflowing with example programs, this book fully describes the techniques necessary to skillfully program, optimize and debug in Turbo C. Every topic and Turbo C feature discussed is fully demonstrated in Turbo C source code examples. Advanced topics such as pointers; direct screen I/O; inline statements in Turbo C; and how to intercept and redirect BIOS calls are all covered in depth. The author further demonstrates these advanced topics by writing a RAM resident pop-up program in Turbo C. In addition, the author fully outlines the differences between Unix C and Turbo C; the transition from Turbo Pascal to Turbo C; and the superset of K&R C features implemented in Turbo C and included in the proposed ANSI C standard.

Book & Disk (MS-DOS)
Item #45-3 $39.95
Book
Item #38-0 $24.95



C Chest and Other C Treasures from Dr. Dobb's Journal
by Allen Holub

This comprehensive anthology contains the popular "C Chest" columns from *Dr. Dobb's Journal of Software Tools*, along with the lively philosophical and practical discussions they inspired, in addition to other information-packed articles by C experts. Topics covered include: pipes, wild-card expansion and quoted arguments; sorting routines; command-line processing; queues and bit maps; utilities such as ls, make, and more; expression parsing; hyphenation; IBM cursor control and an Fget that edits; redirection; accessing IBM video display memory; trees; an AVL tree database package; directory traversal; sets; shrinking .EXE file images; hashing, expressions, and Roman numerals; and statistical applications of digital low-pass filters. Other treasures include: a variable metric minimizer; Fgrep; a peephole optimizer; and curve fitting with cubic splines.
All subroutines and programs are written in C and are available on disk with full source code. MS/DOS format.

Book & Disk (MS-DOS)
Item #49-6 $39.95
Book
Item #40-2 $24.95

SPECIAL OFFER

Receive the **C Chest** book & disk, the **Turbo C** book & disk, and the **Small-Windows** manual & disk, all for only $93.95! You save 15%!

C Chest/Turbo/Window Package
Item #168 $93.95

TO ORDER: Return this order form with your payment to:
M&T Books,
501 Galveston Dr.,
Redwood City, CA 94063
Or, call TOLL-FREE 800-533-4372
Mon-Fri 8AM-5PM
(In CA call 800-356-2002)

In Calif. 800-356-2002

C Disk Formats: Please specify MS-DOS or CP/M. For CP/M, specify: Apple, Kaypro, Osborne, Zenith Z-100 DS/DD, 8" SS/SD

ORDER FORM

Name ____________
Address ____________
City ____________ State ______ Zip ______

Item # Description Price

Subtotal ______
CA residents add sales tax ______ % ______
Add $2.25 per item for shipping ______
TOTAL ______

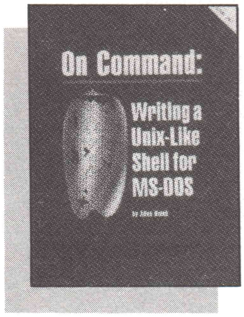
For Disk Orders, please indicate format. Refer to product description for standard format availability.

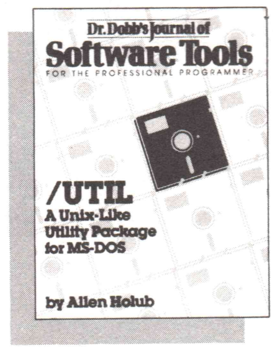
☐ MS-DOS CP/M: ☐ Kaypro ☐ 8" SS/SD ☐ Osborne ☐ Apple ☐ Zenith Z-100 DS/DD

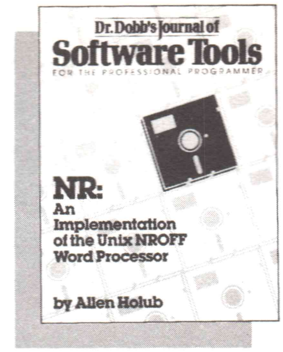
For *Small-Windows*, indicate:
☐ Microsoft version 4.0 compiler ☐ Lattice C 3.0 compiler
☐ Small-C compiler ☐ Turbo C compiler

☐ Check enclosed. Make payable to M&T Publishing.
Charge my ☐ VISA ☐ M/C ☐ Am. Ex.
Card No. ____________ Exp.Date ______

Unix-Like Features for MS-DOS







On Command: Writing a Unix-Like Shell for MS-DOS

by Allen Holub

This book and ready-to-use program demonstrate how to write a Unix-like shell for MS-DOS, with techniques applicable to most other programming languages as well. The book and disk include a detailed description and working version of the Shell, complete C source code, a thorough discussion of low-level MS-DOS interfacing, and significant examples of C programming at the system level.

Supported features include: read, aliases, history, redirection and pipes, Unix-like command syntax, MS-DOS-compatible prompt support, C-Shell-based shell scripts, and a Shell variable that expands to the contents of a file so a program can produce text that is used by Shell scripts. The Unix-like control flow includes: if/then/else; while; foreach; switch/case; break; continue.

The ready-to-use program and all C source code are included on disk. For IBM PC and direct compatibles.

Book & Disk (MS-DOS)
Item #29-1 **$39.95**

\Util

by Allen Holub

When used with the Shell, this collection of utility programs and subroutines provides you with a fully functional subset of the Unix environment. Many of the utilities may also be used independently. You'll find executable versions of cat; cp; date; du; echo; grep; ls; mkdir; mv; p; pause; printevn; rm; rmdir; sub; and chmod.

The ***\Util*** package includes complete source code on disk and all programs (and most of the utility subroutines) are fully documented in a Unix-style manual. For IBM PC and direct compatibles.

Manual & Disk (MS-DOS)
Item #12-7 **$29.95**

NR: An Implementation of the Unix NROFF Word Processor

by Allen Holub

NR is a text formatter that is written in C and compatible with UNIX's NROFF. Complete source code is included in the ***NR*** package so that it can be easily customized to fit your needs. ***NR*** also includes an implementation of the -ms (manuscript) macro package and an in-depth description of how -ms works. NR does hyphenation and simple proportional spacing. It supports automatic table of contents and index generation, automatic footnotes and endnotes, italics, boldface, overstriking, underlining, and left and right margin adjustment. ***NR*** also contains:

- extensive macro and string capability
- number registers in various formats, including Roman and Arabic numerals, both spelled out and in outline form
- diversions and diversion traps (macros that are triggered automatically)
- input and output line traps

NR comes configured for any Diablo-compatible printer, as well as LaserJet and Hewlett Packard's ThinkJet. It is easily configurable for most other printers. Both the ready-to-use program and full source code are included.
For PC compatibles.

Manual & Disk (MS-DOS)
Item #33-X **$29.95**

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT 871 REDWOOD CITY, CA

POSTAGE WILL BE PAID BY ADDRESSEE

M&T Books
501 Galveston Dr.
Redwood City, CA 94063

SPECIAL OFFER

Save 15%

Receive ***On Command, \Util*** and ***NR*** together for only $85.95! You get all the convenience of Unix-like feature and save 15%.

Unix-like Features Package
Item #167 **$85.95**

Vendors

APDA
Apple Programmer's and Developer's Association
290 S.W. 43rd St.
Renton, WA 98055
(800) 426-3667
In WA (800) 527-7562
In Canada (800) 237-4644
(206) 251-5222

Apple Certified Developer Program
Developer Programs
Apple Computer Inc.
20525 Mariani Ave.
Mailstop 27-W
Cupertino, CA 95014
(408) 996-1010

Computer Literacy Bookshops
2590 N. First St.
San Jose, CA 95131
(408) 435-1118
(seven days a week)

Computerware
350 Cambridge Ave.
Palo Alto, CA 94306
(800) 235-1155
In CA (800) 323-1133

Consulair 68000 Development System
Consulair
140 Campo Dr.
Portola Valley, CA 94025
(415) 851-3272
Reader Service No. 29

Enhance Expansion Board
SuperMac Technology
295 North Bernardo
Mountain View, CA 94043
(415) 964-8884
Reader Service No. 30

Fedit Plus
MacMaster Systems
108 E. Fremont Ave., Ste. 37
Sunnyvale, CA 94087
(408) 773-9834
Reader Service No. 31

Fry's Electronics
541 Lakeside Dr.
Sunnyvale, CA 94086
(408) 662-3566

Lightspeed C
Think Technologies
135 South Rd.
Bedford, MA 01730
(800) 648-4465
(617) 275-4800
Reader Service No. 32

MacNosy
Jasik Designs
343 Trenton Wy.
Menlo Park, CA 94025
(415) 322-1386
Reader Service No. 33

MacTutor
MacTutor
P.O. Box 400
Placentia, CA 92670
(714) 630-3730
Reader Service No. 34

QUED/M 2.04
Paragon Concepts Inc.
4954 Sun Valley Rd.
Del Mar, CA 94014
(619) 481-1477
Reader Service No. 35

ResEdit
Available through APDA (see above) or via one of the many on-line services.

Signetics Corp.
Publication Services
Mailstop 27
P.O. Box 3409
Sunnyvale, CA 94088-3409
(408) 991-3620

TMON 2.8
ICOM Simulations Inc.
648 S. Wheeling Rd.
Wheeling, IL 60090
(312) 520-4440
Reader Service No. 36

TO THE MACS
(continued from page 104)

Mail, Delphi (STANKRUTE), and CompuServe (73137,2121) and also by mail at 18617 Camp Creek Rd., Hornbrook, CA 96044—eds.

Bibliography

Apple Computer Inc. *Inside Macintosh*, 4 vols. Reading, Mass.: Addison-Wesley, 1985–1986.

Harbison, Samuel P.; and Steele, Guy L., Jr. *C: A Reference Manual* (2d ed.). Englewood Cliffs, N.J.: Prentice-Hall, 1987.

Knaster, Scott. *How to Write Macintosh Software*. Hasbrouck Heights, N.J.: Hayden, 1986.

Signetics Corp. *Signetics S68000 User's Guide*. Sunnyvale, Calif.: Signetics Corp., 1983.

Smith, David E. ed. *The Best of MacTutor, Volume 1*. Placentia, Calif.: MacTutor, 1986.

Smith, David E. ed. *The Complete MacTutor, Volume 2*. Placentia, Calif.: MacTutor, 1987.

Weston, Dan. *The Complete Book of Macintosh Assembly Language Programming*, 2 vols. Glenview, Ill.: Scott, Foresman, 1986.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

DDJ

(Listings begin on page 54.)

Vote for your favorite feature/article.
Circle Reader Service **No. 4.**

Object-Oriented Programming in Pascal

Object-oriented programming has become quite popular in the last few years. One aspect of its growing acceptance is the increasing number of structured and AI language implementations that support this new paradigm—for example, there are object-oriented implementations in C, Pascal, LISP, Logo, and Forth. In recognition of the magazine's focus this month on the 68000 line, I'll look at a new object-oriented Pascal available for the Macintosh.

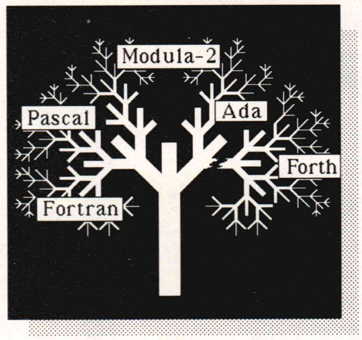
The strength and attraction of object-oriented programming is that it offers you both a new concept (or technique, if you like) for modular coding and a method for efficiently describing a hierarchy of data structures (because redundant components need not be redeclared).

Objects in Pascal can be regarded as highly evolved record structures. Objects declare instance variables (similar to fields of Pascal records) and instance methods, or methods for short. These methods are routines that manipulate the objects. Consider the following object-type declaration:

```
TYPE TRectangle = OBJECT
   { instance variables }
   Length,
   Width : real;
   { instance methods }
   FUNCTION Area : real;
   FUNCTION Circumf : real;
   FUNCTION Diagonal : real;
END;
```

by Namir Clement Shammas

which declares a class of objects named *TRectangle* that contains two instance variables—namely, *Length* and *Width*. Three methods are supplied to calculate the rectangle's area, circumference, and length of diagonal. The declared methods (all of which are functions in the pre-



ceding example) are implicitly *FORWARD* declarations of the routines' headings. The actual code is placed after all the other declarations.

To use the *TRectangle* object type, I must declare a variable and then allocate its dynamic memory using the predefined *NEW()* procedure. Accessing the fields and routines employs the familiar dot notation used with Pascal records. Consider the following code fragment:

```
VAR Rect : TRectangle;
   A : real;

BEGIN
   NEW(Rect);
   Rect.Length := 8.0;
   Rect.Width := 5.0;
   A := Rect.Area;
```

In this code the name of the object is used in conjunction with all the object's instance variables and methods.

A rectangular parallelepiped is a 3-D rectangle that can be regarded as extending the 2-D "parent" shape into a third dimension. This means the parallelepiped shares the same features as the 2-D rectangle and adds a few new ones. To enable the parallelepiped objects to inherit the features of the 2-D rectangle, I declare the following:

```
TYPE TSolid = OBJECT(TRectangle)
   height : real;
   FUNCTION Volume : real;
   FUNCTION Space_Diag : real;
END;
```

This declaration states that the object type *TSolid* is a child of the object type *TRectangle*. As a child object, it is able to inherit all the instance variables and methods of its parent object:

```
VAR Solid : TSolid;
   A, V : real;

BEGIN
   NEW(Solid);
   Solid.Length := 8.0;
   Solid.Width := 5.0;
   Solid.Height := 7.0;
   A := Solid.Area;
   V := Solid.Volume;
```

Notice how the variable *Solid* (of type *TSolid*) is able to access the instance variables and methods of the *TRectangle* object type directly. Unlike nested record structures in Pascal, nested objects can make direct reference to the ancestor's variables and methods.

In the preceding discussion, I pointed out that the class of objects *TSolid* is able to inherit components from its parent object. Object-oriented Pascal provides a convenient mechanism to enable a new subclass of objects to define its own version of certain inherited methods. Such methods offer either additional or alternate ways of manipulating an object.

Consider, for example, the case of making an alternate definition of the object type *TSolid*. I want to rename the method *Space_Diag* as *Diagonal*. Because the *Diagonal* name is also used by a method in object *TRectangle*, I must resolve the conflict I have just created. The keyword *OVERRIDE* is used to override inherited versions, so my new alternate declaration for the *TSolid* object type is:

```
TYPE TSolid = OBJECT(TRectangle)
   height : real;
   FUNCTION Volume : real;
```

FUNCTION Diagonal : real; OVERRIDE;
END;

The *OVERRIDE* keyword is placed after the sought method, followed by a semicolon. To revert to the method inherited from a parent object type, instead of the overridden method, the keyword *INHERITED* must be placed before the method in question.

In certain cases it may be necessary to make a reference to an object associated with a method. Object types are similar to pointers, and hence they must be dynamically allocated. A problem arises because methods cannot allocate the objects they manipulate. To solve this dilemma, object-oriented Pascal provides a special identifier, *SELF*. Using *SELF*, you can make references to the object that is yet to be created—for example, the code for method *Volume* can be written as:

```
FUNCTION Volume : real;
BEGIN
  Volume := SELF.Length *
      SELF.Width *
      SELF.Height;
END;
```

TML Pascal for the Mac, the only object-oriented Pascal implementation I've seen, allows you to omit the *SELF* reference.

A Sample Program

Object-oriented programming can be applied across the board, including with basic data structures. Listing One, page 66, shows a complete TML Pascal program that implements simple objects to represent various types of simple numeric stacks.

The first object defined, *TStack*, contains instance variables and methods used by all the other object types. In this case, the instance variables include the stack height and a Boolean flag used to indicate errors (in the program I use it to indicate that an attempt was made to pop an empty stack; you can also use the flag to indicate an attempt to divide by 0). The methods associated with *TStack* objects initialize, increment, and decrement the stack height.

```
CONST MAX_MENU = 20;

TYPE STRING80 = STRING[80];
     String_Array = ARRAY [0..MAX+MENU] OF STRING80;
     Menu_Range = 0..MAX_MENU;

TMenu = OBJECT
     { declare instance variables }
     Menu_Options : String_Array;
     Num_Options,
     Menu_Choice : Menu_Range;
END;

TItem_Menu = OBJECT(TMenu)
    PROCEDURE Display_Menu;
    FUNCTION Get_Choice : Menu_Range;
END;

TControl_Item_Menu = OBJECT(TMenu);
    Current_Level : Menu_Range;
    PROCEDURE Display_Menu;
    FUNCTION Get_Choice : Menu_Range;
END;

TMain_Pull_Down = OBJECT(TMenu)
    PROCEDURE Display_Menu;
    FUNCTION Get_Choice : Menu_Range;
END;

TPull_Down = OBJECT(TMenu)
    Hot_Key_Char : ARRAY [0.MAX_MENU] OF CHAR;
    Location : ARRAY [0.MAX_MENU] OF INTEGER;
    Attribute : ARRAY [0.MAX_MENU] OF BYTE;
    Active : ARRAY [0.MAX_MENU] OF BOOLEAN;
    PROCEDURE Display_Menu;
    FUNCTION Get_Choice : Menu_Range;
END;
```

Example 1: *Declaration of object types for various kinds of menus*

(continued from page 109)

The second object type, *TRealStack*, is a child of object *TStack*. It defines the stack as a four-element, real-type array. The methods associated with this object type are *Push*, *Pop*, and *Add*. (I've omitted other math-related routines to keep the listing short.) The third type, *THPStack*, is a child of the object type *TRealStack*. This new object type defines the instance variable *LastX* to store the value of the first array element when a stack addition is performed. This also dictates that an overridden *Add* method be defined.

Finally, the fourth object type, *TIntStack*, is a descendant of object *TStack* that implements an integer-type version of object *TRealStack*. The main code portion exercises the methods defined with the objects.

Menus As Objects

Objects find applications in other data structures, such as lists, arrays, and matrices. They can also be applied to data structures representing screens, windows, and menus. Example 1, page 109, shows the declaration for object types representing different kinds of menus.

The *TMenu* object type defines

Object-oriented Pascal limits objects to single inheritance

instance variables that perform the following:

- Tackle the text for menu options.
- Store the number of actual options available.
- Store the number of the option selected (the 0th option is used for exiting from the menu).

TItem_Menu is a menu object type that builds on *TMenu* simply by adding two methods: one to display the menu and another to return the selected choice. The object type *TControl_Item_Menu* is a modified version of *TItem_Menu* that allows you to implement progressive menus that gradually reveal more options. These menus display an itemized menu that contains only the options available to you at the current level.

The *TMain_Pull_Down* object type uses the same instance variables of object type *TMenu*; however, it implements its own version of the methods to display the main menu of a pull-down menu system. The individual options of a pull-down menu are defined by the object type *TPull_Down*. This type declares an additional number of instance variables that are arrays that serve to:

- Define the hot-key characters.
- Locate where the hot-key characters are displayed and their accompanying display attributes.

What experts are saying about PC Scheme from Texas Instruments:

"...Complete, well-designed and inexpensive at $95 ...a complete Lisp implementation for the professional programmer."

PC Tech Journal, August 1986 (Product of the Month)

Discover how powerful—and inexpensive—PC symbolic programming can be with PC Scheme from Texas Instruments. Whether you're an experienced Lisp programmer or just beginning, PC Scheme is the complete, $95* solution to your software development needs.

PC Scheme combines elegant simplicity with remarkable speed in a full Lisp development system. Named *PC Tech Journal*'s Product of the Month (August 1986), PC Scheme brings professional Lisp programming features to personal computers.

PC Scheme 3.0

—Optimizing incremental byte-code compiler for ease of programming and operation
—EMACS-like editor
—Lexical scoping of variables
—Ability to suspend PC Scheme, execute DOS-based programs, then return to PC Scheme
—Random-file access and binary-file support
—Extensions for debugging, graphics and windowing
—External language interface to C, Turbo Pascal® and other languages
—SCOOPS (Scheme Object-Oriented Programming System)
—Two-megabyte extended/expanded memory support
—New manuals with tutorials and examples

Find out for yourself why experts are praising PC Scheme. For the dealer nearest you, or to order by phone, call toll-free:

1-800-527-3500

** TI Suggested list price*

PC Scheme runs on IBM® Personal Computers and compatibles (including the Texas Instruments Business-Pro™ computer). Minimum configuration: 512K RAM, dual floppy system.

TEXAS INSTRUMENTS

261933B

(continued from page 110)

- Define Boolean flags to indicate whether an option is active or passive.

Inheritance Problems

Object-oriented Pascal limits objects to single inheritance: one object type has at most one parent object type. Other languages and implementations, such as Smalltalk, Object Logo, and ExperCommon LISP, support multiple inheritance. I'd like to see multiple inheritance implemented in future versions of object-oriented Pascal because it offers the flexibility and power to model real-world objects realistically. This new level of sophistication generates new problems, though.

Among the first problems to resolve is the question of inheriting instance variables and methods that are present in the ancestor objects. The solution involves assigning influence levels, or priorities, to ancestor objects and providing the ability to override them. I suggest some alternate rules for resolving inheritance conflicts by assigning influence levels as follows:

1. The first ancestor object mentioned is the dominant parent, and all other ancestor objects are of equal importance. In the following example:

TYPE Car = OBJECT(Make, Engine, Body)

the ancestor object *Make* has the greater influence concerning conflicting instance variables and methods and the objects *Engine* and *Body* have the same influence. This syntax assumes that there are no conflicts pending between the last two object types.

2. The order of listing the ancestor objects indicates their influence levels. The following object heading declaration illustrates this syntax:

TYPE Car = OBJECT(Make, Engine, Body)

Here, the *Make* and *Body* object types have the strongest and weakest influences, respectively.

3. The list of ancestor object types is partitioned into two sublists using a special character—say, the bar symbol. The first sublist has its ancestor object types listed in decreasing influence; the second has the rest of the ancestor object types on equal footing. Consider the heading:

TYPE Car = OBJECT(Make, Engine | Body, Doors)

The first sublist contains object types *Make* and *Engine*, and the second sublist contains *Body* and *Doors*. The object type *Make* has the dominant influence, followed by *Engine*. The *Body* and *Doors* object types have an equal influence, which is weaker than that of the first two. This syntax also assumes that there are no conflicts pending between the last two object types.

4. Influence levels are explicitly assigned to all ancestor object types using the syntax <*object type*> = <*unsigned integer constant*>. The integers used need not be in any particular sequence, as shown in the following example:

TYPE Car = OBJECT(Make = 100, Engine = 1, Body = 40)

This object type declaration indicates that the object type *Make* has

the highest influence (by virtue of the number assigned to it and not its position in the list of object types). By contrast, the *Engine* object type is the least influential.

The first and third alternatives contain a common weak point: they may be unable to resolve all the inheritance conflicts. The second and fourth suggested syntaxes are conflict-proof. Nevertheless, you need a mechanism to arbitrate or override inheritance rules explicitly. I suggest using the keyword *FROM* to indicate the parent object type supplying the particular attribute, as shown by the following general syntax:

```
<variable> : data <type> FROM
                    <object type>
<method> FROM <object type>
```

Tracking down inherited variables and methods in multiple inheritance is much more complex than in single inheritance. A single link in the latter is replaced by a complex search graph in multiple inheritance. The increased real time for processing multiple inheritance could be compensated for, however, by using fast CPUs and electronic disks, which would keep the overall compilation time relatively short.

A Final Note

Writing this column for the last few years has been fun, but it has also required a great deal of writing time. Other obligations, including the job of editor of M&T's *Turbo Tech Report*, have made it ncessary for me to pass this column's duties on to another member of the *DDJ* family. Although I will continue to write an occasional article for *DDJ*, this month marks my last Structured Programming column. Thank you for your support, and take care of yourselves.

Availability

All the source code for articles in this issue is available on a single disk. To order, send $14.95 to *Dr. Dobb's Journal*, 501 Galveston Dr., Redwood City, CA 94063, or call (415) 366-3600, ext. 216. Please specify the issue number and format (MS-DOS, Macintosh, Kaypro).

Bibliography

Cox, Brad J. *Object-Oriented Programming: An Evolutionary Approach*. Reading, Mass.: Addison-Wesley, 1987.

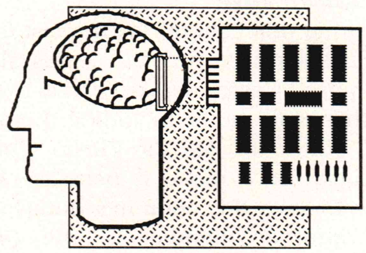
Schmuker, Kurt J. *Object-Oriented Programming for the Macintosh*. Hasbrouck Heights, N.J.: Hayden, 1986.

DDJ

(Listing begins on page 66.)

Vote for your favorite feature/article
Circle Reader Service **No. 5.**

Actor Does More Than Windows



This month I'll be looking at the latest version of The Whitewater Group's object-oriented language Actor. The Whitewater Group has been promoting this language simply as an effective tool for rapid development under Microsoft Windows. It is certainly that. But Actor is also the first object-oriented programming tool specifically designed for writing applications that run under Windows. That fact and The Whitewater Group's intention to port Actor for other windowing systems make it an interesting tool for developing precisely the AI applications that the object-oriented approach is good at while easing the burden of developing for different machines. In this column I'll describe how the latest version (1.1) of Actor implements some of the Microsoft Windows constructs and will examine it from that perspective.

There are so many similarities between Smalltalk and Actor that you may well ask why anyone would develop a new language when implementations of Smalltalk already exist. There are two reasons. First, Actor was designed from the start to be compatible with commercial personal computer environments. Hence its built-in compatibility with Microsoft Windows today and The Whitewater Group's plans to allow window environment programming with Actor across windowing environments tomorrow. Second, the syntax of Actor has been designed to be familiar to C and Pascal programmers. The Whitewater Group's intent was to incorporate all that was good about Smalltalk in a package tailored to the needs of present-day personal computer programmers.

by Ernest R. Tello

To give you a feeling for programming in Actor, I have sprinkled some sample programs through this column as Examples 1 through 4, pages 116, 118, and 125. Three of these are familiar benchmark programs for generating prime and Fibonacci numbers and the Tower of Hanoi and the fourth contains Actor author Chuck Duff's extensions for list processing.

The Actor Desktop

Basically, Actor runs on top of MS-Windows and uses the Windows facilities to implement objects that allow an interactive, windowing environment for development. As with MS-Windows, you can theoretically get along without a mouse, but you wouldn't want to.

The main items used on the Actor desktop are the workspace windows, browsers, and inspectors, which are all modeled on the Smalltalk originals.

When the system first comes up, it shows a workspace window with two rows of command options along the top bar of the window. The commands include *File*, *Edit*, *Doit!*, *Browse!*, *Inspect!*, *Show Room!*, and *Templates*.

The editing area of a workspace window behaves just like an interpreter does. If you type in an expression and then a carriage return, Actor will attempt to compile and execute it. If there is a body of text already in an editing window, then highlighting a portion of that code and clicking on *Doit!* will result in that portion of code being compiled and executed. The *Browse!* and *Inspect!* options result in a new one of these tools opening like a pop-up window on the Actor desktop.

Inspectors are used for focusing on a particular object. You use them to examine the contents of objects in detail as well as to make modifications to them. The upper-left pane of an inspector window contains a list box that displays all the instance variables of an object. Clicking the mouse on any of the items in the scrollable list of variables results in its value being displayed in the bottom pane of the inspector—its edit window. Both class objects and their instances can be accessed using inspectors.

A browser provides a similar function to that of an inspector, but instead of providing an interactive window on a single object, it does this for the entire system of classes. Its scrollable list box contains a list of all the classes currently in the Actor class hierarchy. The right-hand list box contains the methods for the current class. By first selecting a class and then a method in that class, you may access the code in the bottom window and edit it. An Options selection on the browser menu bar allows you to choose whether the classes are listed in hierarchical or alphabetical order.

A Class Act

Actor comes with a large class library of ready-made code that can be used for building applications quickly, once you bridge the learning curve of using the system and knowing what's there. Here is a partial list of the classes used for data structures and the graphics facilities subsumed under them:

```
Object
  Collection
    Indexed Collection
      Array
        Function
        OrderedCollection
          SortedCollection
```

Breakthrough in interface management. Generate C code from Dan Bricklin's Demo screens. Date fields. Full color support. Money fields. Fully programmable field behavior. Scrolling text within fields. Calculator style numeric input. User definable entry validation. Field marking. Orthogonal field movement. Specify fields by number or location. Source code included. Screen sizes limited only by memory. Interfaces with db_VISTA and other libraries. Text style numeric input. Input masking. List fields. Create spreadsheets. Includes Look & Feel screen designer. Integer fields. String formatting commands. Date and time validation functions. Generate C code with Look & Feel screen designer. Supports automatic vertical and horizontal scrolling. Clean screen fields per screen limited only by development. String fields. Easy to painting. Bind as much data as de data entry with commas. Ask a gramming library. Hexadecimal or No fields. Float fields. Quick C. Speaker functions. Lattice. Create Slug. Numeric validation routines. keystroke level. Customize screens 30 day money back guarantee. Gen assortment of editing commands. windows. Assign validation data to credentials. Pull down menus. Sup mode. All functions are kept in C style function reference. Pop-up functions. Numeric range checking. tive function names. Date and time Capture screens from existing as deep as desired. Easy to main checking. Date and time conver definition language based on C's ly definable borders. The current cally highlighted. Create reports.

C-scape 2.0

with

Look & Feel™

NEW!

The state-of-the-art interface management system preferred by professional C programmers and consultants worldwide.

machine. Number of memory. Fast screen modify. Fast screen sired to fields. Numeric bout our linear pro fields. Long fields. Yes Read only fields. reports. Codename Validate data at the and menus at run time. eric data pointer. Rich Easy to learn. Pop-up fields. Corporate C ports EGA 43 line separate modules. Full prompt and message No royalties. Descrip- conversion routines. programs. Nest screens tain. Run time error sion functions. Screen printf. Time fields. Ful- field can be automati- Exploding borders. Convert old programs to C. Borders with titles. Color map enables use of logical colors. Toll-free telephone support line. 24 hour bulletin board. Automatically detects type of monitor being used. ANSI driver included. Screen and field definitions. Uses device drivers for portability. View text in pop-up word entry fields. paging functions Customizable lect different cur Supports CGA, monochrome. cludes functions the display. In ANSI device dri ety of keyboard Lined borders. menuing systems. scroll lights. Vid ver included. drivers can be cre map enables log colors. Borders lines. Fully inte system. Create as as needed. Create screens. Easy to ual. Professional port. Includes

Look & Feel

- WYSIWYG screen design tool
- Generates readable C code
- Create menus and data entry screens
- Define fields of any type
- Variables, prompts, and validation
- Line draw and erase
- Block, move, cut, paste, copy
- Horizontal and vertical scrolling
- Edit Dan Bricklin Demo slides
- Full color support
- Fast, easy, and fun to use
- Includes help
- Full-feature demo available

windows. Read only fields. Rich assortment of editing commands. Pass- Includes ROM BIOS driver. Fields can support any data type. Scrolling/ included. Specify writeable and non-writeable positions within fields. borders. Se sor types. EGA, and Aztec. In for writing to cludes an ver. A vari functions. Multi-level Borders with eo RAM dri New device ated. Color ical use of with prompt grated help many screens data entry follow man customer sup higher level functions. Device drivers swappable at run-time. Context sensitive help system. Cross referenced help screens. Protected fields. Object-oriented design. Read in screen defini tions from disk files. Digitally mastered. Assign prompt strings to fields. UNIX. No run- time license. Numeric range checking. Unified field theory. Full printf % substitution

C-scape 2.0

- Windows, windows, windows
- Menus, menus, menus
- Vast help system
- Create any type of field
- Data entry and validation
- Smart borders
- Extensive function library
- Swappable device drivers
- Easy to learn and use
- Easy to maintain and modify
- Unsurpassed flexibility
- Professional manual
- No royalties; no run-time license
- Source code included
- Demo package available

Oakland Group, Inc.

675 Massachusetts Avenue
Cambridge, MA 02139-3309

800-233-3733 CALL
617-491-7311 NOW

PC/MS-DOS $279, plus shipping (includes C-scape, Look & Feel, source, manual and support). UNIX/others call. 30-day review.

within screen definitions. Supports all memory models. C Bricklin run. Turbo C. 24 hour bulletin board. Higher level functions included. Object-oriented design. All library functions are kept in separate modules. Nest screens as deep as desired. Design screens with Look & Feel screen designer. New device drivers can be created. Create as many screens as needed. No run-time license. Hexadecimal fields. Prefer- red by professionals and consultants. Microsoft. Cross referenced help system. by space aliens. Generate C code with Look & Feel screen designer. Context sensitive help system. Scroll lights. Read in screen definitions from disk files. Automatic vertical and horizontal scrolling. Batteries not included. Double and single line borders. Cross-referenced help system. Save and restore regions of the display. Nested menus. Quick C. Create screens from ASCII files. Easy to learn and use. Horizontal and vertical scrolling. Used by consultants and corporations worldwide. Easy to maintain. Professional documentation. Screen designer creates readable C code. Portable. Easily modifiable functions. No royalties. Source code included. Turn Dan Bricklin slides into C. Professional support. Interface examples for data base management. Validation at keystroke level. Vast integrated and indexed context-sensitive help system. Save and restore regions of the display. Now supporting Quick C, Turbo C, Aztec, Lattice, Microsoft, UNIX and others. And that's not all. Call for demo.

CIRCLE NO. 179 ON READER SERVICE CARD

ARTIFICIAL INTELLIGENCE
(continued from page 114)

```
            TextCollection
        ByteCollection
          String
            Symbol
          Struct
            DosStruct
            GraphicsObject
              Polygon
              Rect
                Ellipse
                RndRect
            Proc
```

Actor Syntax

As with languages such as C and Pascal, Actor encloses arguments to a method in parentheses immediately following the method's name or selector. And like Pascal and Smalltalk, variable assignments are made using the colon-equal (:=) symbol. Also as in Smalltalk, the way you create new instances is by sending the *new* message to a class and assigning this new instance a name. So, for example, you could create an instance of the *Turtle* class by saying:

```
inherit(Object, #Sieve, nil, nil,nil);!!

now(Sieve);!!

/* Returns the number of prime numbers between 0 and cnt,
inclusive. */
Def sieve(self, cnt | flags, count, c)
{       c := cnt + 1;
        flags := new(Array, c);
        fill(flags, true);
        count := 1;
        do( over(2, c),
                { using(i | triple)
                        if flags[i]
                        then triple := i*3;
                                if triple < cnt
                                then  do( overBy(triple-1, c,
i+i-1),
                                  { using(j) flags[j] := nil });
                               endif;
                                count := count + 1;
                        endif;
                });
        ^count;
}!!

Actor[#Sam] := new(Sieve)!!
/* To run type: sieve(Sam, 100) */!!
```

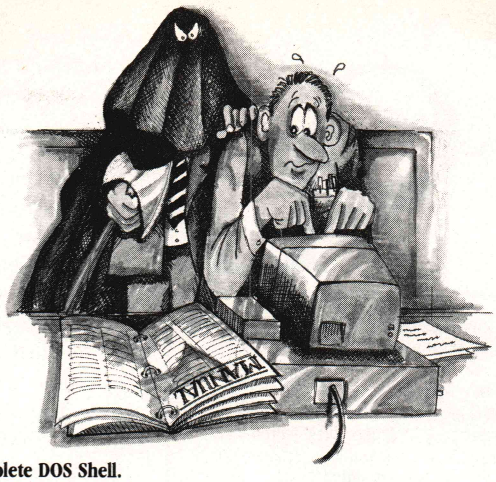
Example 1: *Eratosthenes' Sieve benchmark*

```
now(Int)!!

/* Recursive way of finding the nth Fibonacci term.  Note that
this way of finding the Fibonacci terms is very inefficient
because each message "spawns" two recursive messages.  */

Def fib(self)
{ if self < 3
 then ^1
 endif;  ^fib(self - 1) + fib(self - 2);
}!!

/* Iterative way of finding the nth Fibonacci term. */

Def fib2(self | term, term1Before, term2Before)
{ if self < 3
  then ^1
  else term := 2; term1Before := 1; term2Before := 1;
    do(new(Interval, 3, self + 1, 1),
      {using(i) term := term1Before + term2Before;
       term2Before := term1Before;
       term1Before := term;
      });
    ^term
  endif;
}!!
```

Example 2: *Fibonacci program*

Quit Wasting Time!

As a programmer, most of your time is spent writing and debugging source code, and documenting your work. A powerful, easy-to-use programmable text editor could be saving you HOURS of unnecessary effort.



Only MULTI-EDIT has all these time-saving features:

Fully automatic Windowing and Virtual Memory.
Edit multiple files regardless of physical memory size.
Easy cut-and-paste between files.
View different parts of the same file.

Powerful, EASY-TO-READ high-level macro language.
Standard language syntax.
Full access to ALL Editor functions.
Automate repetitive tasks.
Easy, automatic recording of keystrokes.

Language-specific macros for ALL major languages.
Smart indenting.
Smart brace/parenthesis/block checking.
Template editing.
Supports C, Pascal, BASIC and Assembler.

Terrific word-processing features for all your documentation needs.
Intelligent word-wrap.
Automatic pagination.
Full print formatting with justification, bold type, underlining and centering.
Even a table of contents generator.

Compile within the editor.
Automatically positions cursor at errors.
Built-in MAKE capabilities.
Run compiled program without leaving editor.
Automatically allocates all available memory to compiler or program.

Complete DOS Shell.
Scrollable directory listing.
Copy, Delete and Load multiple files with one command.
Background file printing.

Regular expression search and translate.
Condensed Mode display, for easy viewing of your program structure.
Pop-up FULL-FUNCTION Programmer's Calculator and ASCII chart.

and MOST IMPORTANT,
the BEST user-interface on the market!

- Extensive context-sensitive help.
- Choice of full menu system or logical function key layout.
- Function keys are always labeled on screen (no guessing required!).
- Excellent online, interactive tutorial.
- Keyboard may be easily reconfigured and re-labeled.

Users of Wordstar and Turbo Pascal's Editor could be programming in a fraction of the time with these features.

NO EDITOR ON THE MARKET TODAY HAS ALL THESE FEATURES, OR OFFERS YOU THIS MUCH POWER AT A REASONABLE PRICE, EXCEPT

Multi-Edit $99.

VERSION 2.0

COMPLETE
Or Get our FULLY FUNCTIONAL DEMO Copy for only $10!

	Multi-Edit	BRIEF 2.0	Norton Editor	Vedit Plus
Edit 20+ files larger than memory	Yes	Yes	No	Yes
Powerful high level macro language	Yes	Yes	No	Yes
Full UNDO	Yes	Yes	No	No
Visual marking of blocks	Yes	Yes	Yes	No
Column oriented block operations	Yes	Yes	No	No
Automatic file save	Yes	Yes	No	No
Online help	Extensive	Limited	Limited	Limited
Online tutorial	Yes	No	No	Yes
Choice of keystroke commands or menu system	Yes	No	No	Yes
Function Key assignments labeled on screen	Yes	No	No	No
WP Functions	Extensive	Limited	Limited	Extra Cost
Complete DOS shell	Yes	No	No	No
Pop-up Programmer's Calculator and ASCII table	Yes	No	No	No ASCII
Unlimited 'Off the Cuff' keystroke macros	Yes	No	No	Yes
Allocates all available memory to compiler when run from within editor	Yes	No	Yes	Yes
Intelligent indenting, template editing and brace/parenthesis/block matching and checking for all major languages	Yes	C Only	No	Limited
Flexible condensed mode display	Yes	No	Yes	No
Optional background communications and Spell Checker modules	Yes	No	No	No
PRICE	**$99**	**$195**	**$50**	**$185**

Requires IBM/PC/XT/AT/PS2 or full compatible, 256K RAM, PC/MS-DOS 2.0 or later-Multi-Edit and American Cybernetics are trademarks of American Cybernetics. BRIEF is a trademark of Underware, Inc. Norton Editor is a trademark of Peter Norton Computing, Inc. Vedit is a registered trademark of CompuView Products Inc. Copyright 1987 by American Cybernetics.

Get our FULLY FUNCTIONAL DEMO Copy for only $10!

To Order, Call 24 hours a day:
1-800-221-9280 Ext. 951
In Arizona: 1-602-890-1166
Credit Card and COD orders accepted

American Cybernetics
138 Madrid Plaza
Mesa, AZ 85201

CIRCLE NO. 181 ON READER SERVICE CARD

(continued from page 116)

```
Barney := new(Turtle);
```

In general, sending messages in Actor is like passing arguments but in reverse. The object to which the message is sent is treated as an argument. Once you have created the turtle *Barney*, you can get him to do your bidding by sending various messages that he can recognize. Like any authorized turtle, *Barney* knows that the message *r* means turn right, *l* turn left, *f* move forward, and *b* move backward. He also knows that *down* means to put his tail to the ground for drawing purposes and *up* means to pick it up again.

So, if you wanted *Barney* to perform a turtle walk in the shape of a square, the messages you would send him would be:

```
down(Barney);
f(Barney, 10);
r(Barney, 90);
f(Barney, 10);
r(Barney, 90);
f(Barney, 10);
r(Barney, 90);
f(Barney, 10);
up(Barney);
```

In Actor, the keyword *Def* is used to define methods. So, if you wanted to teach not only *Barney* but also all authorized turtles the new message *walkSquare*, you would define the method:

```
Def walkSquare(self, size)
{ down(self);
  f(self,size);
  r(self, 90);
  f(self, size);
  r(self, 90);
  f(self, size);
  r(self, 90);
  f(self, size);
  up(self);
};
```

Henceforth, to get *Barney* or any of his relatives to perform this maneuver, all you would have to do is to say:

```
walkSquare(Barney, 10);
```

The more astute turtle watchers have probably noticed that this turtle walk is only one orientation for this type of maneuver. There is a species of turtle, admittedly rare, that instinctively will do its square-walking counterclockwise. Fortunately, object-oriented systems such as Actor provide a way of mirroring this little sidelight of natural history. What you can do to cover this complexity is to define a new class of turtle called *CounterTurtle* and provide a *walkSquare* method for turtles

```
/*  ref.  Byte August, 86  p. 146   cbd  8.13.86 */!!

inherit(Object, #TowerOfHanoi, nil, nil, nil);!!

now(TowerOfHanoi);!!

Def moveTower(self, height, from, to, use)
{  if height > 0
   then
     moveTower(self, height - 1, from, use, to);
     moveTower(self, height - 1, use, to, from);
   endif;
}!!

Def moveTower2(self, height, from, to, use)
{  if height > 0
   then
     moveTower(self:TowerOfHanoi, height - 1, from, use, to);
     moveTower(self:TowerOfHanoi, height - 1, use, to, from);
   endif;
}!!

Actor[#Hanoi] := new(TowerOfHanoi);!!

/* Example solves runs the Tower of Hanoi problem */!!
moveTower(Hanoi, 3, 1, 3, 2);!!
```

Example 3: *The Tower of Hanoi*

of this species that does the counterclockwise variant of the standard turtle square walk. Coincidentally, this also provides me with the opportunity of illustrating how new classes are defined in Actor.

The way you usually create new classes in Actor is from within a browser, so I'll do it that way first. Very simply, you first select the class *Turtle* from the class list. Then you go to the Options pull-down menu and select *Make Descendant*. A pop-up window then opens that serves as a template for creating the new class. In this case, let's enter *CounterTurtle* as the name of the class. Now you just click on the Accept button, and the system will create this new class and its name will be added to the class list and become part of the Actor class hierarchy.

The other way to create new classes, which is what the browser is actually doing, is to write the code for it directly. The *inherit* statement is used for this. So, you could write:

```
inherit(Turtle, #CounterTurtle) !!
```

One of the most attractive things about the Actor system is that it provides built-in classes for making the use of the Microsoft Windows user interface easier. Three main classes are concerned with this: *Window*, *Control*, and *ModalDialog*. First let's look at the *Window* class and its descendants. Here is an outline of this branch of the class hierarchy:

```
Object
  Window
      PopupWindow
          ToolWindow
              Browser
              Inspector
      TextWindow
          EditWindow
              WorkEdit
                  BrowEdit
                  FileWindow
                  Workspace
          ScanWindow
          WorkWindow
```

Though relatively large, *Window* is just a formal or abstract class. This means that it implements the methods that will be used by the sub-

(continued from page 119)

classes that implement the specialized windows that actually get instantiated and used. In particular, *Window* implements the routines that communicate with MS-Windows.

The *TextWindow* class is one of the simplest descendants of *Window* that you can get to actually do real things. This class allows you to create tiled windows that can print text. It does this with the *printString* and *printChar* methods, which call the *Textout GDI* (Graphics Display Interface) function in MS-Windows.

Often you will want text windows that can do more that just show text—that can allow you to go in and edit that text. The subclass of *TextWindow* called *EditWindow* provides the code that supports this editing ability. The *WorkEdit* class takes this one step further by allowing you to create windows that can not only edit but can also enter Actor language statements to be evaluated. The three subclasses of *WorkEdit* provide the types of windows that are like those most often used—browsers, file browsers, and general-purpose workspace windows. The main difference, though, is that, like their ancestor *TextWindow*, these are tiled windows.

The windows most often used in Actor come from another branch of the tree that is implemented with the class *PopupWindow*. The windows you get by instantiating *PopupWindow* are the familiar layered windows that stack up on top of one another. Unlike the tiled windows, however, they do not permit you to zoom or contract them down to an icon. As is dictated by MS-Windows, pop-up windows have to have a "parent" text window. When this parent window is contracted into an icon, then the pop-up windows associated with it temporarily become invisible.

Controls and Dialog Boxes

As in MS-Windows, in Actor a control is a special type of window that is used for routine input and output in a user interface. Examples of controls include things like buttons, list boxes, and scroll bars. The branch of the class tree concerned with controls looks like this:

```
Object
   Control
      Button
      ListBox
         ClassList
      Scrollbar
```

Like *Window*, the *Control* class in Actor is just a formal one, and its subclasses are the ones that are actually instantiated in applications. Controls are handled in Actor in an almost identical way to windows. New instances are created by sending the *new* message, and they are displayed by sending the *show* message.

I'll describe one other class—the *ModalDialog* class and its subclasses.

```
Object
   ModalDialog
      ClassDialog
      DebugDialog
      DirtyCLD
      FileDialog
      InputDialog
```

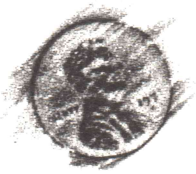
Modal dialog boxes resemble pop-up windows in that they stack on top of other windows and they need a parent window with which they are associated. Like *Window* and *Control*, *ModalDialog* is basically an abstract class that implements code intended for use by its descendants. The *FileDialog* class in Actor is used to create the dialog boxes that routinely appear in MS-Windows when you load a file by using a pull-down menu. The *ClassDialog* class is for dialog objects used when a class is being edited or created with a browser.

Using Actor for AI

For AI applications, as well as many other types of application, processing linked lists is essential. How do you go about doing that in Actor? One way might be to work with the *OrderedCollection* class and add subclasses to it with the necessary methods defined for list processing.

Example 4, a demo provided with the current release of Actor, offers another approach, however. The new class *ListNode* is defined as a subclass of the *Collection* class. The

Remember,



Smart Buying Decisions

Start with DDJ

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #200, DALTON, MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Reader Service Dept.
P.O. Box 507
Dalton, Mass. 01227

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #200, DALTON, MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Dr. Dobb's Journal of Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Reader Service Dept.
P.O. Box 507
Dalton, Mass. 01227

Use this card for FREE, FAST information about the products and services listed in this issue. Simply circle the appropriate numbers below.

Name ______________________________
Title ______________________________
Company ____________________ Phone (___) __________
Address ______________________________
City/State/Zip ______________________________

1	26	51	76	101	126	151	176	201	226	251	276	301	326	351	376
2	27	52	77	102	127	152	177	202	227	252	277	302	327	352	377
3	28	53	78	103	128	153	178	203	228	253	278	303	328	353	378
4	29	54	79	104	129	154	179	204	229	254	279	304	329	354	379
5	30	55	80	105	130	155	180	205	230	255	280	305	330	355	380
6	31	56	81	106	131	156	181	206	231	256	281	306	331	356	381
7	32	57	82	107	132	157	182	207	232	257	282	307	332	357	382
8	33	58	83	108	133	158	183	208	233	258	283	308	333	358	383
9	34	59	84	109	134	159	184	209	234	259	284	309	334	359	384
10	35	60	85	110	135	160	185	210	235	260	285	310	335	360	385
11	36	61	86	111	136	161	186	211	236	261	286	311	336	361	386
12	37	62	87	112	137	162	187	212	237	262	287	312	337	362	387
13	38	63	88	113	138	163	188	213	238	263	288	313	338	363	388
14	39	64	89	114	139	164	189	214	239	264	289	314	339	364	389
15	40	65	90	115	140	165	190	215	240	265	290	315	340	365	390
16	41	66	91	116	141	166	191	216	241	266	291	316	341	366	391
17	42	67	92	117	142	167	192	217	242	267	292	317	342	367	392
18	43	68	93	118	143	168	193	218	243	268	293	318	343	368	393
19	44	69	94	119	144	169	194	219	244	269	294	319	344	369	394
20	45	70	95	120	145	170	195	220	245	270	295	320	345	370	395
21	46	71	96	121	146	171	196	221	246	271	296	321	346	371	396
22	47	72	97	122	147	172	197	222	247	272	297	322	347	372	397
23	48	73	98	123	148	173	198	223	248	273	298	323	348	373	398
24	49	74	99	124	149	174	199	224	249	274	299	324	349	374	399
25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400

January '88: Use before April 30, 1988

1. Did you buy this issue on a newsstand? ( ) Yes ( ) No
2. Are you a subscriber? ( ) Yes ( ) No
3. Have you purchased a product as a result of seeing it advertised in *Dr. Dobb's Journal?* ( ) Yes ( ) No

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

Use this card for FREE, FAST information about the products and services listed in this issue. Simply circle the appropriate numbers below.

Name ______________________________
Title ______________________________
Company ____________________ Phone (___) __________
Address ______________________________
City/State/Zip ______________________________

1	26	51	76	101	126	151	176	201	226	251	276	301	326	351	376
2	27	52	77	102	127	152	177	202	227	252	277	302	327	352	377
3	28	53	78	103	128	153	178	203	228	253	278	303	328	353	378
4	29	54	79	104	129	154	179	204	229	254	279	304	329	354	379
5	30	55	80	105	130	155	180	205	230	255	280	305	330	355	380
6	31	56	81	106	131	156	181	206	231	256	281	306	331	356	381
7	32	57	82	107	132	157	182	207	232	257	282	307	332	357	382
8	33	58	83	108	133	158	183	208	233	258	283	308	333	358	383
9	34	59	84	109	134	159	184	209	234	259	284	309	334	359	384
10	35	60	85	110	135	160	185	210	235	260	285	310	335	360	385
11	36	61	86	111	136	161	186	211	236	261	286	311	336	361	386
12	37	62	87	112	137	162	187	212	237	262	287	312	337	362	387
13	38	63	88	113	138	163	188	213	238	263	288	313	338	363	388
14	39	64	89	114	139	164	189	214	239	264	289	314	339	364	389
15	40	65	90	115	140	165	190	215	240	265	290	315	340	365	390
16	41	66	91	116	141	166	191	216	241	266	291	316	341	366	391
17	42	67	92	117	142	167	192	217	242	267	292	317	342	367	392
18	43	68	93	118	143	168	193	218	243	268	293	318	343	368	393
19	44	69	94	119	144	169	194	219	244	269	294	319	344	369	394
20	45	70	95	120	145	170	195	220	245	270	295	320	345	370	395
21	46	71	96	121	146	171	196	221	246	271	296	321	346	371	396
22	47	72	97	122	147	172	197	222	247	272	297	322	347	372	397
23	48	73	98	123	148	173	198	223	248	273	298	323	348	373	398
24	49	74	99	124	149	174	199	224	249	274	299	324	349	374	399
25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400

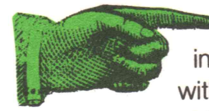
January '88: Use before April 30, 1988

1. Did you buy this issue on a newsstand? ( ) Yes ( ) No
2. Are you a subscriber? ( ) Yes ( ) No
3. Have you purchased a product as a result of seeing it advertised in *Dr. Dobb's Journal?* ( ) Yes ( ) No

Dr. Dobb's Journal of
Software Tools
FOR THE PROFESSIONAL PROGRAMMER

For Free Info ...

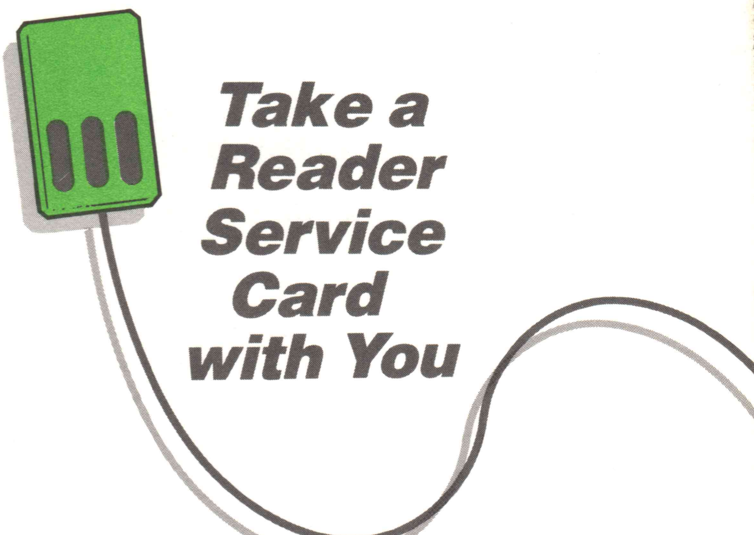
Start Here



Smart buyers start with *DDJ*'s free information card, a shopping center filled with information about the products and services advertised in this very issue: everything from software and systems to peripherals and professional support services.

And smart buyers can use this free information card to quickly and easily gather a comprehensive file of facts, figures and product specs to sort out competing claims. Using *DDJ*'s free information card can prevent you from making the wrong, costly buying decision.

Be a smart shopper. Complete and mail this postage paid card today!



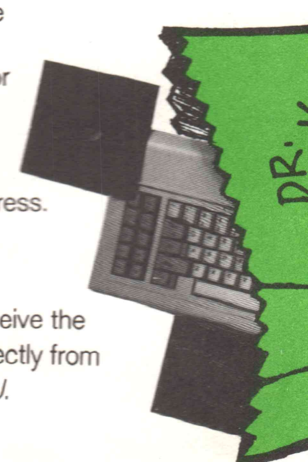
Take a Reader Service Card with You

It's Easy as ...

1. Circle the appropriate free information numbers, referring to the advertiser index for more information.

2. Fill in your name and address.

3. Mail today—postage is absolutely free. You'll receive the product information you need directly from the manufacturers, thanks to *DDJ*.



The Advertiser Index

*The advertiser prefers to be contacted by phone; consult ad.

Advertising Sales Offices

Midwest
Charles Shively (415) 366-3600

Northern California/Northwest
Lisa Boudreau (415) 366-3600

Northeast
Cynthia Zuck (718) 499-9333
Martha Brandt (415) 366-3600

Southern California/AZ/NM/TX
Michael Wiener (415) 366-3600

Telemarketing Rep./SE/SW USA
Cheri Blum (714) 761-0294

Director of Marketing and Advertising
Ferris Ferdon (415) 366-3600

ARTIFICIAL INTELLIGENCE
(continued from page 120)

methods *append, do, isAtom, printOn,* and *rPrintOn* are defined for this new class. New methods, including *isAtom, rPrintOn,* and *cons,* are also defined for the root class, *Object*. If you inspect the code for *cons,* you will see that there is a routine for sending the message *new* to the *ListNode* class and creating a new instance of it. Obviously, this is only the rudimentary beginning of what a functional list-processing class would encompass.

As far as the suitability of Actor for AI applications is concerned, the same limitations that apply to Smalltalk apply here. As compared with LISP and PROLOG, Smalltalk and Actor are relatively low-level languages. They are suitable for developing AI applications, but many additional high-level methods and classes have to be written from scratch just to get started.

The structures used by the *OrderedCollection* class differ significantly from dynamically modifiable linked lists. In the terminology of object-oriented programming, ordered collections are fixed collections that are nevertheless "growable." This means that when you create an *OrderedCollection,* you must create one with a maximum number of elements. If the elements already stored in the collection have not yet reached the maximum, it is easy to add new elements to the beginning or end of the list. When the maximum is reached, and you need more, you must send the *grow* message to the collection. What really happens when you do this is that a new array of the needed size is created and the elements of the old array are copied into it.

Debugging

With Version 1.1 a new debugger has been added to Actor. Currently, both a low-level and high-level debugger are provided, but the low-level debugger is not formally supported and may disappear in later releases of the Actor system.

Routine errors in code evaluated by Actor result in a dialog box that contains a stack history up to the point of the error. The dialog box usually also contains a message that diagnoses the type of error. If you wish, when a dialog box is open because of an error, you can click on the Debug button and cause a Debug window to open.

This is a versatile debugging tool that combines some of the features of a browser and some of those of an inspector as well as the ability to change any of the values associated with a method. With it you can also resume processing on the fly immediately after an error has been fixed.

Conclusions

On the whole, I find Actor to be a thorough implementation of a full programming system with an excellent set of demo programs and helpfully written documentation and tutorials. The ideal users of Actor, as I see it, would be programmers who have already had some exposure to Smalltalk and need to prototype something quickly to run in the MS-Windows environment. For purposes such as these, it is hard to beat.

One thing about the implementation of Actor I dislike, though, is the absence of a facility for multiple inheritance. With systems intended for real-world applications, multiple inheritance should be a standard feature because in the real world many things fulfill multiple roles and multiple functions. Multiple inheritance provides a ready way of handling this in an explicit way.

Vendors of object-oriented tools who fail to include multiple inheritance typically say that they do not want to make the system too complex for users or that none of their customers have requested it. I have not found either of these explanations at all convincing. I have had no difficulty in using multiple inheritance in systems that have it and can't imagine trying to build a serious object-oriented application without it.

In response to this, it might be said that you can still create classes of the same definition in a system without multiple inheritance the hard way simply by defining them to be exactly what you want. My feeling about this is that it may be true in theory but it tends to be something that is never done in prac-

ARTIFICIAL INTELLIGENCE
(continued from page 122)

tice. In the two years or so that I have worked with object-oriented programming systems, I have never done this unless the system provided for multiple inheritance, in which case it becomes a routine practice.

The reason is the same one that makes interactive systems such as Actor significant and not just a mere convenience. The more you make basic things easy to do, the more you tend to launch out into the more difficult areas creatively, trying things that otherwise you might not have tried. If you are a user of an object-oriented programming tool that lacks this feature, I strongly recommend that you pester the vendor to put it at the top of the list for new features. I feel almost certain that you will not regret the results of having exercised your prerogative as a customer.

Version 1.1 of Actor differs from 1.0 in two main ways. First of all, there is more space—an additional 70K—for compiling applications. Second, it is fully compatible with Windows II, though it does not fully support all of Windows II's facilities. A future release of Actor, though, will actually support programming with the new features of Windows II. Needless to say, the implementations of Actor that will use the full features of Windows II and the OS/2 Presentation Manager will ultimately determine the fate of this product. But if the current implementation is any indication, the future versions should be of very high quality indeed.

DDJ

Vote for your favorite feature/article.
Circle Reader Service **No. 6.**

Vendor

Actor
The Whitewater Group
Technology Innovation Center
906 University Pl.
Evanston, IL 60201
(312) 491-2370
Reader Service No. 28

```
 C.B.Duff  7.13.86
 (c) Copyright, 1986
 The Whitewater Group
 Technology Innovation Center
 906 University Place
 Evanston, Il. 60201  (312) 491-2370        */


now(NilClass);!!
/*  append nil to a node  */
Def append(self, aNode)
{ ^aNode }!!

Def cons(self, aNode)
{ ^aNode }!!

Def rPrintOn(self, aStrm)
{ printOn('[', aStrm);
}!!

inherit(Collection, #ListNode, #(left right), nil,nil);!!

now(ListNode);!!

Def append(self, aNode)
{ ^cons( left, append(right, aNode));
}!!

Def do(self, aBlock)
{ if isAtom(left)
 then eval(aBlock, left);
 else do(left, aBlock);
 endif;
 if right
 then do(right, aBlock);
 endif;
}!!

Def isAtom(self)
{ ^nil }!!

Def printOn(self, aStrm)
{ printOn('[', aStrm);
 printOn(left, aStrm);
 rPrintOn(right, aStrm);
}!!

Def rPrintOn(self, aStrm)
{ printOn(' ', aStrm);  printOn(left, aStrm);
 rPrintOn(right, aStrm);
 printOn(']', aStrm);
}!!

now(Object);!!

Def isAtom(self)
{ }!!

Def rPrintOn(self, aStrm)
{ printOn('.', aStrm);
 printOn(' ', aStrm);
 printOn(self, aStrm);
 printOn(']', aStrm);
}!!

Def cons(self, aNode | newNode)
{ if isAtom(aNode)
 then newNode := new(ListNode);
 else newNode := copy(aNode);
 endif;
 newNode.left := self;
 newNode.right := aNode;
 ^newNode;
}!!
```

Example 4: *List-handling support in Actor*

(continued from page 12)

is a defining word *UNITS*:

```
: UNITS CREATE SWAP , , DOES>
                         D@ */;
```

Then you would have what appears in Example 1, page 12. This is an improvement, but Forth permits a simple addition that allows conversion back to the input units when outputting (see Example 2, page 12).

Obvious modifications include use of floating point (to prevent the unintentional truncations of integer arithmetic) and defining a machine-code sequence *<DO-UNITS>* to replace the run-time instructions following *DOES>* if greater speed is needed. (It probably isn't—these definitions are intended to I/O a relatively small set of measurements from the console or a file.)

Finally, let me put in a plug for HS/FORTH (for IBM PCs/clones): Jim Callahan's Harvard Softworks (P.O. Box 69, Springfield, OH 45066) has been extremely supportive with upgrades, fixes, technical advice, and so on. The version of Forth is extremely fast "as is" but comes with all sorts of goodies and tools for using the 8087 chip, string handling, windowing, graphics, sound, and so on. Although the documentation of several years ago was rather cryptic, it is now excellent and includes Kelly and Spies' textbook. I have never seen this product reviewed or mentioned and cannot imagine why, considering its excellence.

Julian V. Noble
105 Powhatan Cir.
Charlottesville, VA 22901

Instruction Timings

Dear *DDJ*,

I hate to be the bearer of bad news, but Tom Disque's "8088 Assembly-Language Programming Techniques" in the July 1987 issue is inaccurate. In several cases Disque presents instruction timings that are not consistent with his assumptions.

In the second paragraph, Disque says, "Please note that all timings are for the 8088 microprocessor and assume that the 4-byte 8088 prefetch queue is empty at the start of execution." Although this is a good assumption because the 8088 is so common and the prefetch queue is always empty except for a few relatively rare circumstances, the timings that Disque gives for his instruction sequences do not correspond to this statement.

Disque claims that the simple sequence:

```
mov ax, cs
mov es, ax
```

takes four cycles to execute. If the prefetch queue is empty, however, the instructions must be fetched first. These two instructions are 4 bytes, and the 8088 requires four clock cycles per memory cycle. This alone comes to 4×4, or 16 clock cycles, just to fetch the instructions, never mind any additional execution time. If the prefetch queue were full, you would have a different story, but Disque assumed that it wasn't.

I suspect that Disque probably meant to do better than this because later he gives the example:

```
shl ax, 1
shl ax, 1
shl ax, 1
shl ax, 1
```

He says: "Faster (8 cycles [plus 8 more to fetch the two extra instructions] . . . ." I'm not sure what to make of this. Either he meant to assume the prefetch queue was empty or that it was full. In any case, it would take 16 clock cycles to fetch two instructions, not 8 (4 clock cycles per byte times 2 bytes per instruction times 2 instructions).

Let's examine how the 8088 really executes the previous sequence. First, it must fetch the first instruction—eight clocks. It then starts fetching the second instruction while processing the first one. Processing the instruction takes two clocks and fetching the second instruction takes eight. Because eight is more than two, you really have to wait eight clock cycles. At this point the cycle repeats. The total "execution" time is 32 clock cycles.

As you can see, it takes the 8088

• DataWindows major update available
• OS/2 versions

Avoid extra steps.

You've got better things to do than repeat the same steps. Over . . . and over . . . and over. Up your productivity with Greenleaf Software.

With more than 70 new functions added to our popular libraries, Greenleaf is now the most complete and mature C language function resource available. It's no wonder we've been rated the best. Winning program developers in major corporations such as IBM, EDS and GM have proven our reliability in thousands of applications.

Step Lively

New Greenleaf Functions v.3.10 includes 295 of the functions you've been asking for — DOS, disk, video, color text and graphics, string, time/date, keyboard, plus many more! With Greenleaf, you'll finish faster.

Cut Corners

When it comes to merging information, the new Greenleaf Comm Library v.2.10 is the fastest communications facility of its kind. Over 120 functions — ring buffered, interrupt-driven asynchronous communications. And, only Greenleaf gives you the power to build a 16-port communication system.

Get on the Fast Track

Order your new Greenleaf library today! See your dealer or call 1-800-523-9830.

Greenleaf Comm Library	$185.00
Greenleaf Functions	$185.00
Greenleaf DataWindows	$225.00
Greenleaf C Sampler	$ 94.50
Digiboard Comm4	$325.00
Digiboard Comm8	$535.00

In stock, shipped next day.

Greenleaf DataWindows and Turbo C

DataWindows, the finest C programming windows tool available, puts windows, transaction data entry and menus at your fingertips.

Our new TURBO C versions are ready to get you going fast! And our new 3-in-1 C Sampler for only $94.50 supports Turbo C or Quick C with comm, windows, menus and more! Our libraries support all popular C compilers for MS DOS.

GREENLEAF Software

Call Toll Free:
800-523-9830

In Texas and Alaska:
214-248-2561

Greenleaf Software, Inc.
16479 Dallas Parkway, Suite 570
Dallas, Texas 75248

(continued from page 126)

much longer to fetch its instructions than to execute them, so the 8088's speed is proportional to the number of bytes it must transfer. This includes instruction bytes as well as data processed by a program. This also means that the 8088's 4-byte queue is almost always empty.

The few exceptions are multiply and divide instructions, which take longer to execute than to fetch. There are also a few esoteric instructions that take longer to execute than to fetch, but their use is so rare that you can forget about them.

Therefore, to time 8088 code that does not include multiply or divide instructions, all you have to do is add the number of bytes of instruction that must be fetched to the number of bytes of data transferred in memory. Once you have the total number of reads and writes to memory, you multiply by four clocks per fetch. You need no instruction timing tables or photographic memory. All you need to be able to do is count and multiply by 4 (two left shifts in binary).

This same concept is used to time Z80 code, except for one quirk. The Z80 takes four clocks for the first byte of an instruction fetch and three clocks to fetch the data.

What about the 8086? The 8086 can fetch instructions faster than the 8088 can but is still basically limited to how fast it can fetch instructions (and read/write to memory). The 80386 should finally provide enough memory width (32 bits) to make a prefetch queue worthwhile.

Disque redeems himself, however, with his Example 8, which demonstrates an excellent, fast, general-purpose, memory move routine. The correction for odd address transfers shows good understanding of that aspect of the 8086 processor.

Good assembly-language programmers such as Disque are rare, but sometimes we all become victims of Intel's overly optimistic instruction timings. Disque's instruction timings sound like they came from Intel.

Lee Pelletier
5 Burley St.
Wenham, MA 01984

Correction

Dear *DDJ*,

I'd like to provide some additional information about the Async AppleTalk article published in the October 1987 issue of *DDJ*.

Async AppleTalk is based on the source code of Apple Computer's AppleTalk driver. The following notice was dropped from the article during the editing process: The AppleTalk protocols and computer programs are licensed from Apple Computer Inc. and AppleTalk, Macintosh, and LaserWriter are trademarks of Apple Computer Inc. Portions copyright (c) 1985 Apple Computer and copyright (c) 1987 the Trustees of Dartmouth College.

Also, the disk being distributed by *DDJ* does not contain the executable desk accessory. This, along with the source files and other debugging

tools, is available from the authors. Please send a blank (800K) Macintosh disk in a mailer to me at the address listed below.

Richard Brown
Dartmouth College
Async AppleTalk Distribution
Kiewit Computer Center
Hanover, NH 03755

Stonecutter Error

Dear *DDJ*,

With reference to the cover of your September 1987 issue, the statement *TAKAGUCK* will never be executed because *ZOK-OOT-UK* always sets *GLOOTBLEE* false. The correct statement should be *XOK-OOT-OK*, which uses *GLOOTBLEE* as an error indicator. Please inform your stonecutter to proofread the algorithms before printing.

James Sturdevant
Minneapolis, MN

To List or Not to List

Dear *DDJ*,

In response to your question about source listings in Running Light (November 1987), I personally feel a big value in *DDJ* is the source listings. I use them and I'm sure a lot of the silent majority do too. You're right about a typing drill, so I normally try to download instead to typing. I agree with posting them on CompuServe, and maybe the idea of a BBS run by *DDJ* would be a good addition. Most of the bigger PC publications have established one for their readers.

One different idea I would like to push is the use of Cauzin Softstrips from Cauzin Systems ([203] 573-0150) in each publication, like *PC Magazine* does. Cauzin readers are cheap ($200) and will pay for themselves to readers like me who are far removed from California. I thought you were using Softstrips at one time, and I'm not sure what happened. Anyway, it is a cheap alternative for avid software users like me.

I hope you will consider my suggestion. It would sure be a help to a lot of us.

David Doss
Computer Science Dept.
Illinois State University
133 B Stevenson Hall
Normal, IL 61761

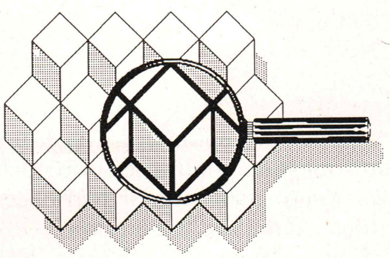
Dear *DDJ*,

I have a few comments regarding your question about the source listings. Keep small (one or two pages) listings. Put long listings on a BBS and make them available on disk. CompuServe is no solution. *PC Magazine*, *Byte*, and *Computer Language* all have great BBSs for downloading. Also, make sure all necessary *include* files are present for C programs. Most published C source code is useless because the author uses routines hidden in header files that are not provided.

Edgar T. Lynk 2G
70 Park Terrace East
New York, NY 10034

DDJ

OF INTEREST



Language Products for the Mac

Two new versions of LPA MacPROLOG are available from **Logic Programming Associates**: the Student Edition uses a built-in incremental compiler and a high-powered declarative graphics environment; the Wizard Edition also has a built-in incremental compiler and high-powered declarative graphics environment along with a new C and Pascal interface, serial I/O, and an optimizing compiler. Both versions require 1 megabyte of memory. The Student Edition costs $275, and the Wizard Edition costs $495. Reader Service No. 16.

Logic Programming Associates
Studio 4
The Royal Victoria Patriotic Building
Trinity Rd.
London SW18 3SX
England
01-8711-2016

Version 2.0 of True BASIC, a language system for the Macintosh and the Macintosh II developed by **True BASIC Inc.**, is now shipping. True BASIC 2.0 offers several new features, including enhancements to the editor; new debugging tools; enhanced speed; and modules, a programming concept found in Modula-2 and Ada. True BASIC supports color graphics for the Mac II as well as the 68881 math coprocessor. Programs written in True BASIC on the Mac can also run on the IBM PC, Amiga, and Atari ST. The retail price of True BASIC 2.0 is $99.95. Owners of previous versions can upgrade to the new version at a cost based on a sliding scale, depending on the date of original purchase. Reader Service No. 17.

True BASIC Inc.
39 South Main St.
Hanover, NH 03755
(604) 643-3882

TurboGeometry Library, a new software tool for Macintosh programmers who write graphics, engineering, educational, and other programs that use geometry, has been released by **Disk Software**. The library comes with source code for more than 150 routines, including routines that find the intersection of lines, polygons, arcs, and planes; determine the coefficients of the equations of lines, circles, arcs, and planes; convert the coefficients of one equation to another; find the distance between points, lines, circles, arcs, and planes; decompose a concave polygon into a series of convex polygons; and more. The library sells for $99.95. Reader Service No. 18.

Disk Software Inc.
2116 East Arapaho, Ste. 487
Richardson, TX 75081
(214) 423-7288

Mac AI

ExperTelligence is now shipping two AI products for the Macintosh: ExperCommonOPS5 and ExperCommon LISP Foreign Function Interface.

Some of the features of ExperCommonOPS5 include a forward-chaining, rule-based, expert system language; a custom tracing mechanism of rule firings and changes to memory; the ability to pause, examine memory, and undo rule firings to revert to an earlier state; conformation to the de facto OPS5 standard described in *Programming Expert Systems Using OPS5* (Brownston et al., Addison-Wesley); complete portability to other implementations of OPS5; facilities to interface with other programming languages and tools; the ability to create complex Macintosh user interfaces for developed expert systems; and integration with the most powerful AI development environments available today. ExperCommonOPS5 runs in both ExperCommon LISP for the Macintosh Plus and SE and ExperCommon LISP II for the Macintosh II, Version 2.2 or later. The retail price is $595.

The ExperCommon LISP Foreign Function Interface is an add-on component of the complete ExperProfessional Development System. Users can create a new version of the ExperCommon LISP language that contains functions written in any Macintosh Programmer's Workshop language: C, Pascal, or assembly language. Once the new version has been created, the foreign functions can be called from ExperCommon LISP in the same way as any other built-in function. The Foreign Function Interface runs in ExperCommon LISP on the Macintosh Plus and the Macintosh SE and runs on the Macintosh II with ExperCommon LISP II. The price is $295. Reader Service No. 19.

ExperTelligence Inc.
559 San Ysidro Rd.
Santa Barbara, CA 93108
(805) 969-7874

Human Intellect Systems has announced Instant-Expert Plus, a Macintosh expert system shell. Instant-Expert Plus uses natural-language rule entry, interactive graphics, variables, and more than 18 different inference engine search strategies. Hardware requirements are a Macintosh 512K with an external drive. Instant-Expert Plus retails for $498. Reader Service No. 20.

Human Intellect Systems
1670 S. Amphlett Blvd., Ste. 326
San Mateo, CA 94402
(415) 571-5939

For the Atari ST and Amiga

A new version of the Metacomco Toolkit is now available from **Metacomco**. The toolkit consists of 11 useful AmigaDOS commands that are stored in the C directory and can be used in the same way as the standard DOS commands. The new version also includes a Unix-based make utility and a touch utility. Commands included in the package are Pipes, Librarian, Disassembler, Auxil-

iary CLI, Mount, Browse, Enlarge, Pack, Unpack, Make, and Touch. Metacomco Toolkit 1.2 works under Versions 1.1 and 1.2 of AmigaDOS and can augment the Metacomco shell. The price for the package is $49.95. Reader Service No. 21.
Metacomco
26 Portland Sq.
Bristol BS2 8RZ
England
44-272-438781

FTL Modula-2 for the Atari ST from **Workman & Associates** is now shipping. The Atari version is fully compatible with existing FTL programs and includes a complete GEM interface. The package sells for $79.95. Reader Service No. 22.
Workman & Associates
1925 East Mountain St.
Pasadena, CA 91104
(818) 791-7979

Miscellaneous

MEMOCOM is now shipping a universal cross-development kit for the Macintosh that includes a table-driven cross-assembler and a MEMULATOR II or MEMULATOR 16 in-circuit EPROM emulator. With the kit, developers can use the Macintosh to assemble and test sources programs for virtually any microprocessor/controller with 24 or less address bits. Both the universal cross-assembler and MEMULATOR II/16 support industry-standard Intel hex, Motorola S-record, and straight binary formats. These output file formats are compatible with most serial EPROM programs. The MAC Universal Cross-Development Kit sells for $725 with a MEMULATOR II and $1,275 with a MEMULATOR 16. The kit is also available for PC-DOS and MS-DOS systems and the Atari ST. Reader Service No. 23.
MEMOCOM
1920 Arbor Creek Dr.
Carrollton, TX 75010
(214) 446-9906

Addison-Wesley has released the Programmer's Online Companion by Steven Capps, a disk-based database reference to *Inside Macintosh*, Volumes 1–4, and the *Apple Numerics Manual*. The companion is a utility program that allows programmers immediate access to bits of information from within any language development system and includes almost all the system calls, system globals, and assembly-language equates found in those volumes. The program controller resides in 5K RAM. The structure assumes a basic knowledge of Pascal or assembly language, *Inside Macintosh*, and the Macintosh itself. The package includes a 3.5-inch disk and 32 pages of documentation. It runs on the Macintosh 128K, Macintosh 512K, Macintosh 512Ke, and Macintosh Plus and sells for $34.95. Reader Service No. 24.
Addison-Wesley Publishing Co.
Rte. 128
Reading, MA 01867
(617) 944-3700

Whitesmiths has announced CXDB, an interactive C source-level cross-debugger for the Motorola MC680x0 line of processors. CXDB is a host-resident debugger available for the VAX and IBM PC that allows embedded 68K programs to be debugged in terms of the original C source

(continued from page 131)

code. Features allow users to execute host system commands; print the contents of a C variable function or file with or without assembly-language display; step through single or multiple lines until a specific location is reached, again with or without assembly-language display; create log output for later examination; set and remove breakpoints on a specific C variable, line, or function; disassemble C source lines; obtain in-line help; display stack frames with or without type; and value information for variables in each frame. Prices for CXDB range from $1,800 to $7,000. Reader Service No. 25.
Whitesmiths Ltd.
59 Power Rd.
Westford, MA 01886
(617) 692-7800

Optotech has introduced Laser DataBank, a "plug-and-play" 5¼-inch write-once read-many (WORM) optical drive subsystem for four popular computer environments: Sun Microsystems' Sun-3; DEC's MicroVAX II; IBM's PC, PC/XT, and 286/386-based computers, including the PS/2 and compatibles; and Apple's Macintosh SE and Macintosh II. The proprietary software also allows data exchange among all four subsystems by writing disks that can be read by any Optotech drive. All the subsystems are built around the Optotech Model 5984 Optical Disk Drive with 400 megabytes of storage per 5¼-inch optical disk drive. Proprietary interface software in each subsystem makes the write-once drive immediately compatible with the host computer's operating system and provides standard file and disk management functions.

The price for the Macintosh Laser DataBank is $3,995. Single-unit prices for the Sun and VAX are $5,950 and $6,950, respectively. The Laser DataBank for PCs is currently available at a single-unit price of $2,995. Reader Service No. 26.
Optotech Inc.
740–770 Wooten Rd.
Colorado Springs, CO 80915-3518
(303) 570-7500

Reduce, from **Northwest Computer Algorithms**, is an interactive software system designed for general mathematical computations of interest to scientists, mathematicians, engineers, and university students. The system has been applied to a variety of problems in many different research areas, including quantum electrodynamics, quantum chromodynamics, electrical network analysis, plasma physics, celestial mechanics, general relativity, numerical analysis, and a variety of engineering problems such as turbine and ship hull design. The code is portable across a wide range of machines. Reduce is now available for the following machines: VAX; Sun; Apollo; Macintosh; and IBM PC, PC/XT, PC/AT, and compatibles. A single-machine license costs $495. Reader Service No. 27.
Northwest Computer Algorithms
P.O. Box 1747
Novato, CA 94948
(415) 897-1302

DDJ

Upgrade your technology

The software technology available to programmers of IBM-compatible personal computers is truly amazing. And newer, more powerful development packages appear all the time. But until now, finding out about these important products has been a difficult and time consuming task.

FREE Buyer's Guide. The New Programmer's Connection Buyers Guide contains individual descriptions of over 500 titles of programmer's development software by over 150 manufacturers. Each description covers major product features as well as any software or hardware requirements and version numbers. In the box on the right are some examples of the types of descriptions you'll find in our Buyer's Guide.

No Hidden Charges. The low discount prices in our Buyer's Guide are all you pay. We don't charge extra for domestic UPS Ground shipping, credit cards, COD orders, purchase orders, sales tax (except Ohio) or special handling (except for non-Canadian international orders).

Guarantees. We offer FREE 30-day no-risk return guarantees and 30-day evaluation periods on most of our products.

Latest Versions. The products we carry are the latest versions and come with the same manufacturer's technical support as if buying direct.

Large Inventory. We have one of the largest inventories of programmer's development products in the industry. Most orders are shipped within 24 hours.

Noncommissioned Staff. Our courteous salespeople are always ready to help you. And if you aren't sure about your needs, our knowledgeable technical people can give you sound, objective advice.

Experience. We've specialized in development software for IBM-compatible personal computers since 1984 and are experienced at providing a full range of quality products and customer services.

How to Get Your Copy. There are three ways for you to receive your FREE copy of the Programmer's Connection Buyer's Guide: 1) Use the reader service card provided by this journal; 2) Mail us a card or letter with your name and address; or 3) Call one of our convenient toll free telephone numbers.

If you haven't yet received your Programmer's Connection Buyer's Guide, act now. Upgrading your programming technology could be one of the wisest and most profitable decisions you'll ever make.

Limited Time Only!

Sale Prices on ISAM File Managers

through 1/31/88

FairCom

c-tree in C Source Code
List $395 Reg $315 **Sale $289**

c-tree with r-tree
List $650 Reg $519 **Sale $499**

These fast and highly portable B+Tree functions provide multi-key ISAM file management for C programs. There are low level functions for directly accessing data and index files and high level functions for creating and manipulating ISAM files. The highly portable C source code can be compiled with almost any C compiler or computer for single-user, multi-user or network applications. It supports: record locking for multi-users; fixed and variable length records; fixed and variable length keys with key compression; re-use of deleted record space; duplicate and unique key fields; and more. The package includes a complete family of setup and maintenance utilities, unlimited technical support, no royalties, and free hardcopy listings of release updates. r-tree is an optional report generation utility for c-tree that permits complex, multi-line reports to be produced from single or multiple c-tree data files.

Supports all commercial grade C compilers. Requires 128K memory. Version 4.1F.

Lattice

dBC III Plus
List $750 Reg $594 **Sale $499**

With Library Source
List $1500 Reg $1184 **Sale $998**

Use the Lattice dBC III Plus library of functions to write fast C language programs to create, access and update files that are compatible with Ashton-Tate's dBASE III PLUS database management system. dBC III Plus is network ready with functions that solve complicated network database problems. These functions let you lock files or records automatically or manually, prevent you from accidentally locking files or records that are already locked, and allow you to test whether files or records are locked or free. You can share your ISAM files with as many stations as are possible on your network.

Specify compiler (current version): Borland Turbo C, Lattice C, or Microsoft C. Requires 128K memory. Version 1.0.

SoftCraft

Btrieve
List $245 Reg $184 **Sale $169**

Xtrieve
List $245 Reg $184 **Sale $169**

Report Option
List $145 Reg $99 **Sale $89**

Btrieve/N
List $595 Reg $454 **Sale $429**

Xtrieve/N
List $595 Reg $454 **Sale $429**

Report Option/N
List $345 Reg $269 **Sale $249**

Btrieve is a keyed, indexed file management system for use with most programming languages. Btrieve allows a file structure with: record length up to 4K bytes (64K in some environments); up to 24 different keys per file; maximum key size of 255 bytes; a maximum file size of over 4 billion bytes; and file size limited only by physical storage capacity and operating system limitations. Duplicate, modifiable, null and segmented keys are allowed and there is no limit to the number of files open at one time. Written in 8088 assembly language for maximum efficiency, Btrieve uses extensive cache buffering to optimize performance and pre-imaging to automatically recover damaged files. Transaction bracketing and automatic record locking allow you to guarantee the integrity of your data files despite the concurrency problems that arise in a network. The optional Xtrieve is a menu-driven query system that enables Btrieve users to access Btrieve files without writing a program. The Report Option for Xtrieve allows you to easily generate reports.

Specify single-user or multi-user/network version. For multi-user/network version, specify environment: 3Com3plus; IBM TopView; Microsoft Windows; Multilink Advanced; MPOS; Novell Advanced NetWare; XENIX System V/AT; or satellite or server-based IBM PC Network. Btrieve supports most language compilers and interpreters. Requires hard disk and 128K memory (Btrieve routines use 32K). Version 4.10.

CALL TOLL FREE

USA:	800-336-1166
Canada:	800-225-1166
Ohio & Alaska (Collect):	216-494-3781
International:	216-494-3781
Telex:	9102406879
Easylink:	62806530

Programmer's Connection
7249 Whipple Avenue NW
North Canton, OH 44720

PROGRAMMER'S QUALITY INTEGRITY Service SUPPORT PRICE CONNECTION

The Free Programmer's Connection Buyer's Guide.

CIRCLE NO. 198 ON READER SERVICE CARD

ai - expert systems

	List	Ours
1st-CLASS *by Programs in Motion*	495	399
EXSYS *Development Software by EXSYS*	395	289
EXSYS Runtime System	600	469
LEVEL5 *by Information Builders*	685	569
Logic-Line Series *All varieties by Thunderstone*	CALL	CALL
VP-EXPERT *by Paperback Software* *New*	125	79

ai - lisp language

	List	Ours
muLISP-87 Interpreter *by Soft Warehouse* *New*	250	CALL
muLISP-87 Interpreter & Compiler *New*	350	CALL
Q'Nial *Various by NIAL Systems*	CALL	CALL
Star Sapphire LISP *Compiler by Sapiens*	495	429
TransLISP PLUS *from Solution Systems*	195	125

ai - prolog language

	List	Ours
Arity Combination Package	1095	979
Expert System Development Pkg	295	229
File Interchange Toolkit	50	44
PROLOG Compiler & Interpreter	650	569
Screen Design Toolkit	50	44
SQL Development Package	295	229
Arity PROLOG Interpreter	295	229
Arity Standard Prolog	95	77
LPA microPROLOG *All Varieties*	CALL	CALL
MPROLOG Language Primer *LOGICWARE*	50	45
MPROLOG P500 *by LOGICWARE*	495	395
MPROLOG P550 *w/Primer by LOGICWARE*	220	175
Turbo PROLOG *by Borland Intl*	100	64
Turbo PROLOG Toolbox *by Borland Intl*	100	64

ai - smalltalk language

	List	Ours
Smalltalk/V	100	84
EGA/VGA Color Option	50	45
Goodies Diskette #1	50	45
Smalltalk/Comm	50	45

ai - texas instruments

	List	Ours
Arborist *Decision Tree Software*	595	519
PC Scheme *Lisp*	95	77
Personal Consultant Easy	495	435
Personal Consultant Image	495	435
Personal Consultant Online	995	869
Personal Consultant Plus	2950	2589
Personal Consultant Runtime	95	84

ada language

	List	Ours
AdaVantage *GSA-validated by Meridian Software*	795	735
AdaVantage Utility Packages	50	47
DOS Environment Package	50	47
Ada *GSA-validated w/maintenance by alsys* *New*	3355	3119
Janus/ADA C Pak *by R&R Software*	95	84

apl language

	List	Ours
APL*PLUS PC *by STSC*	695	495
APL*PLUS PC Spreadsheet Mgr *by STSC*	195	139
APL*PLUS Tools *by STSC*	295	199
APL*PLUS PS/2 *by STSC*	695	495
ATLAS*GRAPHICS *by STSC*	450	329
Financial/Statistical Library *by STSC*	275	189
Pocket APL *by STSC*	95	69
STATGRAPHICS *by STSC*	895	649

assembly language

	List	Ours
386 ASM/LINK *Cross Asm by Phar Lap*	495	389
8088 Assembler *w/Z-80 Translator by 2500 AD*	100	89
ASMLIB *Function Library by BCSoft*	149	125
asmTREE *B-Tree Dev System by BCSoft*	395	329
Cross Assemblers *Various by 2500 AD*	CALL	CALL
EZASM *by C Source*	70	59
Microsoft Macro Assembler	150	CALL
Resident-ASM *by American Software Intl* *New*	150	139
Turbo Debugger *by Speedware*	89	79
Turbo Editasm *by Speedware*	99	84
Visible Computer: 8088 *by Software Masters*	80	64

basic language

	List	Ours
db/Lib *for QuickBASIC by AJS Publishing*	139	119
Finally *by Komputerwerk*	99	85
MACH 2 *by Micro Help*	69	55
Microsoft QuickBASIC *New Version*	99	CALL
QBase *Relational Database by Crescent*	89	79
Quick-Tools *by BCSoft*	130	109
QuickPak *by Crescent Software*	69	59
Scientific Subroutine Library *by Wiley*	125	99
Screen Sculptor *by Software Bottling*	125	91
Stay-Res *by MicroHelp*	69	55
True BASIC	100	79
True BASIC *w/Run-time* *Special Price*	150	99
True BASIC 3D Graphics	50	41
True BASIC Developer's Toolkit	50	41
Turbo BASIC Compiler *by Borland Intl*	100	64

blaise products

	List	Ours
ASYNCH MANAGER *Specify C or Pascal*	175	135
C TOOLS PLUS/5.0	129	99
KeyPlayer *Super Batch Program*	50	45
LIGHT TOOLS *for Datalight C*	100	65
PASCAL TOOLS & TOOLS 2	175	135
RUNOFF *Text Formatter*	50	45
TURBO ASYNCH PLUS	129	99
TURBO C TOOLS	129	99
TURBO POWER TOOLS PLUS	129	99
VIEW MANAGER *Specify C or Pascal*	275	199

borland products

	List	Ours
EUREKA *Equation Solver*	167	105
Reflex: The Analyst	150	99
Sidekick	85	57
Superkey	100	64
Turbo Basic Compiler	100	64
Turbo Basic Database Toolbox	100	64
Turbo Basic Editor Toolbox	100	64
Turbo Basic Telecom Toolbox	100	64
Turbo C Compiler *(Call for support products)*	100	64
Turbo Lightning and Word Wizard	150	94
Turbo Lightning	100	64
Turbo Lightning Word Wizard	70	47
Turbo Pascal	100	64
Turbo Pascal Database Toolbox	100	64
Turbo Pascal Developer's Toolkit	395	289
Turbo Pascal Editor Toolbox	100	64
Turbo Pascal Gameworks Toolbox	100	64
Turbo Pascal Graphix Toolbox	100	64
Turbo Pascal Numerical Methods Toolbox	100	64
Turbo Pascal Tutor	70	41
Turbo Prolog Compiler	100	64
Turbo Prolog Toolbox	100	64

c language

	List	Ours
C-terp *by Gimpel, Specify compiler*	298	219
C Trainer *with Book by Catalytix*	122	87
DeSmet C *w/Debugger & Large case*	209	184
DeSmet C *w/Debugger only*	159	138
Eco-C88 Modeling Compiler *by Ecosoft* *New*	100	79
Instant C *by Rational Systems*	495	369
Lattice C Compiler *vers. 3.2 from Lattice*	500	265
Mark Williams Let's C *with FREE csd*	75	54
Microsoft C Compiler 5.0 w/CodeView	450	CALL
Microsoft QuickC Compiler	99	CALL
Optimum-C *by Datalight*	139	95
Turbo C Compiler *by Borland*	100	64
Uniware 68000/10/20 Cross Compiler *by SDS*	995	829

c utilities

	List	Ours
Blackstar C Library *by Sterling Castle*	125	98
C++ *by Guidelines w/version 1.1 kernel*	195	172
c-tree & r-tree Combo *by FairCom*	650	499
c-tree *ISAM File Manager*	395	289
r-tree *Report Generator*	295	239
Csharp Realtime Toolkit *by Systems Guild*	600	489
Curses Window Dev Pkg *by Aspen Scientific*	119	105
with Source Code	289	249
dBx *dBASE to C Translator by Desktop AI*	350	299
with Source Code	550	419
Entelekon Combo Package *Special Price*	200	125
Flash-up Windows *by Software Bottling*	90	78
GraphiC *Color version by Sci Endeavors*	350	274
HALO *Graphics by Media Cybernetics*	300	205
HALO Development Pkg *for Microsoft*	595	389
The HAMMER *by OES Systems*	195	129
HOOPS *3-D Graphics by Ithaca Software* *New*	575	459
LINK&LOCATE *by Systems & Software* *New*	350	309
PANEL/TC *for Turbo C by Roundhill*	129	95
PANEL Plus *by Roundhill*	495	395
PC Lint *by Gimpel Software*	139	99
Resident-C *by American Software Intl* *New*	150	139
RTC PLUS *Fortran to C by Cobalt Blue*	450	369
Sapiens V8 *Virtual Memory Manager*	300	265
Scientific Subroutine Library *by Wiley*	175	135
TE Developer's Kit *by Sub Systems*	95	85
Vitamin C *by Creative Programming*	225	149
VC Screen *Forms Designer*	100	79
WKS LIBRARY *by Tenon Software* *New*	89	79
Zview *by Data Mgmt Consultants*	245	119

cobol language

	List	Ours
COBOL spII *by Flexus*	395	329
FPLIB *for Realia COBOL by BCSoft*	149	129
Micro Focus COBOL *See Micro Focus Section*		
Microsoft COBOL *See Microsoft Section*		
Realia COBOL with RealMENU	1145	899
Realia COBOL	995	783
RealCICS	995	783
RM/COBOL *by Ryan-McFarland*	950	CALL
RM/COBOL 85 *by Ryan-McFarland*	1250	CALL
RM/NET+5 *by Ryan-McFarland*	300	CALL
RM/Screens	395	CALL
SCREENIO *by Norcom*	400	379
screenplay *for COBOL by Flexus*	175	115

database management

	List	Ours
Advanced DBMaster *by Macon Software* *New*	510	419
Clipper *by Nantucket* *New*	695	CALL
dBASE III Plus *by Ashton-Tate* *New*	695	389
dbIII Compiler *by WordTech Systems* *New*	CALL	CALL
dbSQL *by WordTech Systems* *New*	CALL	CALL
dbxl *by WordTech Systems* *New*	169	109
dFLOW *by Wallsoft* *New*	149	119
The Documenter *by Wallsoft* *New*	295	239
Fox Base Plus *New*	395	249
Genifer *by Bytel* *New*	395	275
MAGIC PC *by AKER* *Special Price, New*	199	179
Paradox 1.1 *by Ansa/Borland* *New*	495	359
Paradox 2.0 *by Ansa/Borland* *New*	725	525
Paradox Network Pack *by Ansa/Borland* *New*	995	725
Q&A *by Symantec* *New*	349	219
Quickcode *by Fox & Geller* *New*	295	179
Quickindex *by Fox & Geller* *New*	149	95
Quickreport *by Fox & Geller* *New*	295	179
QuickSilver *by WordTech Systems* *New*	599	349
R:Base 5000 *by Microrim* *New*	495	CALL
R:Base System V *by Microrim* *New*	700	CALL
\rdb *by Robinson-Shafer-Wright* *New*	139	119
Advanced Revelation *by COSMOS* *New*	950	CALL
SQL Base *by Gupta Technologies* *New*	995	CALL
Multi-User Version *New*	1995	CALL
Tom Rettig's Library *by Tom Rettig & Assoc* *New*	100	79
UI Programmer *by Wallsoft* *New*	295	239
VP-INFO *by Paperback Software* *New*	125	79
VP-PLANNER *by Paperback Software* *New*	100	63
VP-PLANNER PLUS *by Paperback Software* *New*	125	79
XDB II *by Software Systems Technology* *New*	395	CALL
C Programming Interface *New*	295	CALL

debuggers & profilers

	List	Ours
386 DEBUG *Cross Debugger by Phar Lap*	195	129
Advanced Trace-86 *by Morgan Computing*	175	115
Codesmith-86 *by Visual Age*	145	98
DSD87 *by Soft Advances*	125	79
MiniProbe *by Atron*	395	369
Periscope I *with Board by Periscope*	345	275
Periscope II *with NMI Breakout Switch*	175	139
Periscope II-X *Software only*	145	105
Periscope III *8 MHz version*	995	795
Periscope III *10 MHz version*	1095	875
The PROFILER *with Source Code by DWB*	125	89
TURBOsmith *Source debugger for Turbo Pascal*	99	69
The WATCHER *Profiler by Stony Brook*	60	51

disk utilities

	List	Ours
Back-It *by Gazelle Systems*	130	115
Disk Optimizer *by Softlogic Systems*	60	55
Disk Technician *by Prime Solutions*	100	89
FASTBACK *by 5th Generation Systems*	159	115
FASTBACK PLUS *by 5th Generation Systems* *New*	189	139
United Software Security *Take Two Manager* *New*	139	119
Vcache *by Golden Bow Systems*	50	47
Vopt *by Golden Bow Systems*	50	47
Vfeature *by Golden Bow Systems*	80	74
Vfeature Deluxe *by Golden Bow Systems*	120	111
XenoCopy-PC *by XenoSoft*	80	69

dos utilities

	List	Ours
Advanced Norton Utilities	150	89
Command Plus *by ESP Software*	80	69
Desqview *from Quarterdeck*	130	115
FANSI-CONSOLE *by Hersey Micro*	75	62
Mace Utilities *Paul Mace Software*	99	89
MicroHelp Utility *by MicroHelp*	59	49
Norton Commander *by Peter Norton*	75	55
Norton Utilities *by Peter Norton*	100	59
OPAL *Shell Language by Software Factory*	99	89
Q-DOS II *by Gazelle Systems*	70	59
Taskview *by Sunny Hill Software*	80	55

essential products

	List	Ours
Breakout Debugger *by Essential Software*	125	89
C Utility Library	185	119
Essential Communications	185	125
Essential Communications *with Break Out*	250	189
Essential Graphics	250	183
/*residentC*/ *New*	99	85
with Source Code *New*	198	169
ScreenStar *New*	99	85
with Library Source Code *New*	198	169

forth language

	List	Ours
CFORTH *Native Code Compiler by LMI*	300	229
FORTH/83 Metacompiler *Specify Target*	750	599
MMS Forth *by Miller Microcomputer Svcs* *New*	180	159
PC/FORTH *by Laboratory Microsystems*	150	109
PC/FORTH+ *by Laboratory Microsystems*	250	199
Programmer's Package #1 *by LMI*	250	199
Programmer's Package #2 *by LMI*	350	279
Programmer's Package #3 *by LMI*	500	399
UR/FORTH *Also Available for OS/2 by LMI*	350	279
UR/FORTH Libraries	500	395

fortran language

	List	Ours
50 MORE: FORTRAN *by Peerless Scientific*	125	95
ACS Time Series *by Alpha Computer Service*	495	389
AUTOMATED PROGRAMMER *by KGK Automated*	995	949
Essential Graphics *by Essential Software*	250	183
Forlib-Plus *by Alpha Computer Service*	70	44
FORTRAN Addenda *by Impulse Engr*	165	138
HALO *Graphics by Media Cybernetics*	300	205
I/O PRO *w/No Limit Library by MEF*	250	219
Microcompatibles Combo Package	240	215
Grafmatic	135	117
Plotmatic	135	117
Microsoft FORTRAN *w/CodeView*	450	CALL
No Limit Library *by MEF Environmental*	129	109
Numerical Analyst *by MAGUS*	295	199
PANEL *by Roundhill Computer Systems*	295	199
RM/FORTRAN *by Ryan-McFarland*	599	399
Scientific Subroutine Library *by Wiley*	175	135
Statistician *by Alpha Computer Service*	295	235
STATLIB.GL: *by Wiley*	295	239
STATLIB.TSF: *by Wiley*	295	239
Strings & Things *by Alpha Computer Service*	70	45

greenleaf products

	List	Ours
Greenleaf C Sampler *Specify QuickC or Turbo C*	95	69
Greenleaf Comm Library	185	125
Greenleaf Data Windows Library	225	155
with Source Code	395	249
Greenleaf Functions	185	125

help utilities

	List	Ours
HELP/Control *by MDS*	125	99
On-line Help *from Opt-Tech*	149	99
SoftScreen/HELP *by Dialectic Systems*	195	149

lattice products

	List	Ours
Lattice C Compiler *ver 3.2 from Lattice*	500	265
with Library Source Code	900	495
C Cross Reference Generator	50	37
with Source Code	200	139
C-Food Smorgasbord *Function Library*	150	95
with Source Code	300	179
C-Sprite *Source Level Debugger*	175	CALL
Curses *Screen Manager*	125	85
with Source Code	250	169
dBC III	250	169
with Source Code	500	356
dBC III Plus	750	499
with Source Code	1500	998
LMK *Make Facility*	195	138
RPG II Combo *All three items below*	1100	875
RPG II Compiler *No Royalties*	750	625
SEU *Source Entry Utility*	250	199
Sort/Merge	250	199
Screen Design Aid Utility *for RPG II*	350	309
SecretDisk II *Encryption Utility*	79	59
SideTalk *Resident Communications*	120	88
SSP/PC *Scientific Subroutine Library*	350	269
Text Management Utilities	120	88

metagraphics products

	List	Ours
FontWINDOW	95	79
LightWINDOW/C *for Datalight C*	95	79
MetaWINDOW *No Royalties*	195	159
MetaWINDOW/PLUS	275	229
TurboWINDOW/C *for Turbo C*	95	79
TurboWINDOW/Pascal *for Turbo Pascal*	95	79

CIRCLE NO. 198 ON READER SERVICE CARD

micro focus products

Micro Focus COBOL/2	900	729
Micro Focus COBOL/IQ *Ad hoc Report Writer*	495	395
Micro Focus COBOL/IQ *for DOS 3.X Networks*	995	795
Micro Focus FORMS-2	295	235
Micro Focus Level II COBOL *w/Animator*	495	395
Level II COBOL	349	279
Level II Animator	195	155
Micro Focus PC-CICS	1495	1189
with Micro/SPF	1595	1269
Micro Focus Personal COBOL	149	119
Micro Focus Professional COBOL	CALL	CALL
Micro Focus SOURCEWRITER	995	795
Micro Focus VS COBOL/XENIX	1495	1195

microport products

386 Unlimited License Kit	249	209
AT Unlimited License Kit	249	209
DOSMerge286 *Specify 2-Users or Unlimited*	149	129
DOSMerge386 *2-Users*	395	345
DOSMerge386 *Unlimited Users*	495	429
System V/386 Combination	799	669
386 Runtime System	199	169
386 Software Development System	499	429
Text Preparation System	199	169
System V/AT Combination	549	465
AT Runtime System	199	169
AT Software Development System	249	209
Text Preparation System	199	169
	395	295

microsoft products

Microsoft BASIC Compiler *for XENIX*	695	CALL
Microsoft BASIC Interpreter *for XENIX*	350	CALL
Microsoft C Compiler 5.0 *w/CodeView*	450	CALL
Microsoft COBOL Compiler *with COBOL Tools*	700	CALL
for XENIX	995	CALL
Microsoft Excel *New*	495	CALL
Microsoft FORTRAN *Optimizing Compiler*	450	CALL
Microsoft FORTRAN *for XENIX*	695	CALL
Microsoft Learning DOS	50	CALL
Microsoft Macro Assembler	150	CALL
Microsoft Mouse *Specify Serial or Bus*	150	CALL
with Microsoft Windows	200	CALL
with EasyCAD	175	CALL
Microsoft Pascal Compiler	300	CALL
for XENIX	695	CALL
Microsoft QuickBASIC	99	CALL
Microsoft QuickC	99	CALL
Microsoft Windows	99	CALL
Microsoft Windows 386 *New*	195	CALL
Microsoft Windows Development Kit	500	CALL
Microsoft Word	450	CALL
Microsoft Works *New*	195	CALL

mks products

MKS AWK	75	65
MKS RCS *Revision Control System*	189	155
MKS Toolkit *with MKS VI Editor*	139	109
MKS Trilogy *with AWK, CRYPT & Korn Shell*	119	99
MKS VI *Editor by MKS*	75	65

modula-2 language

LOGITECH Modula-2 Development System	249	199
Modula-2 Compiler Pack	99	79
Modula-2 Toolkit	169	139
LOGITECH Modula-2 Window Pkg	49	39
Macro2 *Macro preprocessor by PMI*	89	79
ModBase *by PMI*	89	79
ModGraph *by TEQNA*	50	45
MODULA-2 *by Stony Brook* *New*	195	169
MODULA-2 with Utilities *by Stony Brook* *New*	345	299
Repertoire *by PMI*	89	75
Science & Engrg Tools *by Quinn-Curtis*	75	67
Universal Graphics Library *by Quinn-Curtis*	130	119

mouse products

LOGIMOUSE BUS *with PLUS Pkg by LOGITECH*	119	98
with PLUS & PC Paintbrush	149	119
with PLUS & CAD Software	189	153
with PLUS & CAD & Paint	219	179
with PLUS & First Publisher	CALL	CALL
LOGIMOUSE C7 *with PLUS Pkg, Specify Connector*	119	98
with PLUS & PC Paintbrush	149	119
with PLUS & CAD Software	189	153
with PLUS & CAD & Paint	219	179
with PLUS & First Publisher	CALL	CALL
Microsoft Mouse *See Microsoft Section*		

other languages

ACTOR *by Whitewater Group*	495	419
CCS MUMPS *All varieties by MGlobal*	CALL	CALL
Marshal Pascal *by Marshal Language Systems*	189	155
Pascal-2 *by Oregon Software*	395	289
Personal REXX *by Mansfield Software*	125	99
SNOBOL4+ *by Catspaw*	95	80

other products

Carbon Copy Plus *by Meridian Technology*	195	159
Dan Bricklin's Demo Pgm *by Software Garden*	75	57
Dan Bricklin's Demo Tutorial	50	45
Fast Forward *by Mark Williams*	70	59
Instant Replay *by Nostradamus*	150	CALL
MicroTEX *Typesetting from Addison-Wesley*	295	CALL
Printer Drivers	CALL	CALL
muMATH *by Soft Warehouse* *New*	300	199
Net-Tools *by BCSoft*	149	129
Norton Guides *Specify Language*	100	65
OPT-Tech Sort *by Opt-Tech Data Proc*	149	99
PC-MOS/386 *by The Software Link* *New*	195	179
PC/TOOLS *by Custom Software Systems*	49	45
Resident Expert *Specify lang by Santa Rita* *New*	59	55
Screen Machine *by MicroHelp*	79	59
SuperSort *by LifeStyle*	139	119

phoenix products

Pasm86 *Macro Assembler version 2.0*	195	108
Pdisk *Hard Disk & Backup Utility*	145	99
Pfantasy Pac *Phoenix Combo*	995	595
Pfinish *Execution Profiler*	395	209
Pfix86plus *Symbolic Debugger*	395	209
PforCe *Specify C Compiler*	395	209
PforCe++ *Specify C Compiler and C++*	395	209
Plink86plus *Overlay Linker*	495	275
Pmaker *Make Utility*	125	78
Pmate *Macro Text Editor*	195	108
Pre-C *Lint Utility*	295	154
Ptel *Binary File Transfer Program*	195	108

polytron products

PolyBoost *Software Accelerator* *Special Price*	80	54
PolyDesk III	99	72
PolyDesk III Archivist	50	42
PolyDesk III Cryptographer	50	42
PolyDesk III Talk	70	52
PolyLibrarian *Library Manager*	99	89
PolyLibrarian II *Library Manager*	149	129
PolyMake *UNIX-like Make Facility*	149	129
PolyShell *Special Price*	149	105
Polytron C Beautifier	50	45
PolyXREF *Complete Cross Ref Utility*	219	185
PolyXREF *One language only*	129	109
PVCS Corporate *Version Control System*	395	329
PVCS Personal	149	129

program mgmt utilities

Interactive EASYFLOW *by Haventree*	150	125
PrintQ *by Software Directions*	89	84
Quilt Computing Combo *QMake & SRMS*	250	199
Sapiens MAKE	179	155
Sapiens MAKE & V8	439	379
Source Print *by Aldebaran Labs*	97	75
TLIB *Version Control System by Burton*	100	89
Tree Diagrammer *by Aldebaran Labs*	77	67

sco products

Complete XENIX System V *by SCO*	1295	994
Development System	595	499
Operating System *Specify XT or AT*	595	499
Text Processing Package	195	144
Lyrix *by SCO*	595	449
SCO Professional *1-2-3 Workalike for XENIX*	795	595
SCO XENIX-NET	595	495
XENIX System V 386 *by SCO*	CALL	CALL

softcraft products

Btrieve *ISAM Mgr with No Royalties*	245	169
Xtrieve *Query Utility*	245	169
Report Option *for Xtrieve*	145	89
Btrieve/N *for Networks*	595	429
Xtrieve/N	595	429
Report Option/N *for Xtrieve/N*	345	249
XQL *New*	795	CALL

text editors

Brief & dBrief Combo *from Solution Systems*	275	CALL
Brief	195	CALL
dBrief *Customizes Brief for dBASE III*	95	CALL
de *by David Livshin*	75	65
Epsilon *Emacs-like editor by Lugaru*	195	147
KEDIT *by Mansfield Software*	125	98
Micro/SPF *by PHASER SYSTEMS*	175	139
Microsoft Word	450	CALL
PC/VI *by Custom Software Systems*	149	99
SPF/PC *by Command Technology Corp*	CALL	CALL
Vedit Plus *by CompuView*	185	128

turbo pascal utilities

ALICE *Interpreter by Software Channels*	95	66
AZATAR DOS Toolkit *by AZATAR* *New*	95	85
DOS/BIOS & Mouse Tools *by Quinn-Curtis*	75	67
Flash-up *by Software Bottling* *New*	89	78
Flash-up Developer's Toolbox *New*	49	45
MACH 2 *for Turbo Pascal by Micro Help*	69	55
MetraByte D/A Tools *by Quinn-Curtis*	100	89
Science & Engrg Tools *by Quinn-Curtis*	75	67
Screen Sculptor *by Software Bottling*	125	91
Speed Screen *by Software Bottling*	35	32
System Builder *by Royal American*	150	129
IMPEX *Query Utility*	100	89
Report Builder	130	115
TDebugPLUS *by TurboPower Software*	60	49
Tmark *by Tangent Designs*	80	69
Turbo Professional *from TurboPower* *New Version*	99	79
TurboHALO *from IMSI*	95	75
TurboPower Utilities *by TurboPower*	95	78
TurboRef *by Gracon Services*	50	35
TURBOsmith *Source Debugger by Visual Age*	99	69
Universal Graphics Library *by Quinn-Curtis*	130	119

wendin products

Operating System Toolbox	99	79
PCNX *Operating system*	99	79
PCVMS *Similar to VAX/VMS*	99	79
Wendin-DOS *Multitasking DOS*	99	85
XTC *Text Editor w/Pascal source*	99	75

xenix/unix products

Btrieve *ISAM File Mgr by SoftCraft*	595	429
C-terp *by Gimpel*	498	379
c-tree *ISAM Mgr by FairCom*	395	289
dBx *with Library Source by Desktop AI*	550	419
DIRECTORY SHELL 286 *by American Mgmt Sys*	349	295
DIRECTORY SHELL 386 *by American Mgmt Sys*	495	415
Epsilon *Text Editor by Lugaru* *New*	195	147
Micro Focus Products *See Micro Focus Section*		
Microport Products *See Microport Section*		
Microsoft Products *See Microsoft Section*		
PANEL Plus *by Roundhill Computer Systems*	795	535
REAL-TOOLS *Binary Version by PCT*	99	89
Complete Source Version	999	729
RM/COBOL *by Ryan-McFarland*	1250	949
RM/FORTRAN *by Ryan-McFarland*	750	549
SCO Products *See SCO Section*		

Terms are subject to change.

LOWEST PRICES

Due to printing lead times, some of our current prices may differ from those shown here. Call for latest pricing.

FREE SHIPPING

Orders within the USA (including Alaska & Hawaii) are shipped FREE via UPS. Express shipping is available at the shipping carrier's standard rate with no rush fees or handling charges. To avoid delays when ordering by mail, please call first to determine the exact cost of express shipping.

CREDIT CARDS

VISA and MasterCard are accepted at no extra cost. Your card is charged when your order is shipped. Mail orders please include credit card expiration date and authorized signature.

CODs AND POs

CODs and Purchase Orders are accepted at no extra cost. No personal checks are accepted on COD orders. POs with net 30-day terms (with initial minimum order of $100) are available to qualified US accounts only.

SALES TAX

Orders outside of Ohio are not charged state sales tax. Ohio customers please add 6% Ohio tax or provide proof of tax-exemption.

INTERNATIONAL ORDERS

Shipping charges for International and Canadian orders are based on the shipping carrier's standard rate. Since rates vary between carriers, please call or write for the exact cost. International orders (except Canada), please include an additional $10 for export preparation. All payments must be made with US funds drawn on a US bank. Please include your telephone number when ordering by mail. Due to government regulations, we cannot ship to all countries.

VOLUME ORDERS

Volume orders may qualify for additional discounts. Call us for special pricing.

SOUND ADVICE

Our knowledgeable technical staff can answer technical questions, assist in comparing products and send you detailed product information tailored to your needs.

30-DAY GUARANTEE

Most of our products (excluding books) come with a 30-day documentation evaluation period or a 30-day return guarantee. Please note that some manufacturers restrict us from offering guarantees on their products. Call for more information.

MAIL ORDERS

Please include your telephone number on all mail orders. Be sure to specify computer, operating system and any applicable compiler or hardware interface(s). Send mail orders to:

Programmer's Connection
7249 Whipple Ave. NW
North Canton, OH 44720

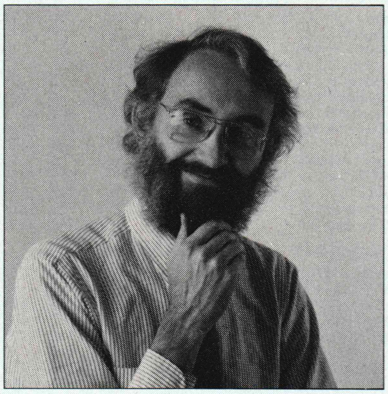
USA	**800-336-1166**
CANADA	**800-225-1166**
OHIO & ALASKA (Collect)	**216-494-3781**
TELEX	**9102406879**
EASYLINK	**62806530**
INTERNATIONAL	**216-494-3781**
CUSTOMER SERVICE	**216-494-8899**

Hours: Weekdays 8:30 AM to 8:00 PM EST.

PROGRAMMER'S QUALITY INTEGRITY Service SUPPORT PRICE CONNECTION

CIRCLE NO. 198 ON READER SERVICE CARD

SWAINE'S FLAMES



It will be seen that this mere painstaking burrower and grub-worm of a poor devil of a sub-sub appears to have gone through the long Vaticans and street-stalls of the earth, picking up whatever allusions to whales he could anyways find in any book whatsoever, sacred or profane. Therefore you must not, in every case at least, take the higgledy-piggledy whale statements, however authentic, in these extracts, for veritable gospel cetology.

—Herman Melville.

Ahab, standing on the deck of Pequod and searching for the great white whale, overlooked the real monster.

Ben, an editor for another magazine, recently moved to the West Coast. Drawn by the California curls, he went out to walk the waves. Last week the ocean drove his surfboard into his face.

It's been nearly seven years since I moved to the Coast, but I still walk out on the rocks regularly to stand in awe of the big blue beast.

At last fall's Comdex, IBM rolled in with an announcement of the delivery of OS/2 and word that it will be promoting its proprietary Extended Edition of OS/2 over the Standard Edition. The Comdex daily declared the show Very Blue, while several frolicking whales spouted their support.

Among the whales, Lotus Development Corp. announced Agenda. Agenda is the information manager that Jerry Kaplan went to Lotus to develop. It is what he calls an item/category database: the items are short, free-form, textual entities, which the user can assign to points in a user-defined, evolving hierarchy of categories. Agenda could be wildly successful if it sufficiently models the näive view of computers held by many pre-users: that one should be able to type arbitrary information into the machine and retrieve it on demand.

Ashton-Tate's chairman, Ed Esber, had left no doubt as to who would define the dBASE standard when he dared dBASE add-on and compiler vendors to "make his day," which they could do by challenging A-T's claim on the dBASE language (the language itself, not just A-T's implementation). So it came as no surprise to see Marty Winston of Wallsoft at Comdex handing out make-my-day buttons decorated with the international red-circle-and-backslash negation symbol. (It was Winston who brought dBASE aftermarket companies together in a dBASE Standards Committee.) Can a language definition be a product? Esber has not only thrown down the gauntlet to the aftermarket companies but also challenged the industry to deal with this tricky question.

Despite the lukewarm response to IBM's RT, you can find RISC technology implementations for just about any broad-based hardware. Shortly before Comdex, Sun announced its SPARC (Scalable Processor ARChitecture) RISC architecture and by Comdex had licensed the technology to over 40 companies, including Arete. Products based on the Inmos transputer parallel RISC processors are also proliferating. Atari, which incidentally would have won any Most Outrageous Product Name at Comdex competition with its Moses Promiselan, demonstrated a prototype of the Abaq transputer, which can act as a backend for an Atari ST and provide mondo MIPS number crunching and near-photographic graphics. Levco, now a division of Scientific Micro Systems, announced the formation of a TransLink Transputer Developers Group to support developers using its transputer modules in Macs.

Comdex offered a few new implementations of familiar languages. Ryan-McFarland, now a division of Austec, announced an OS/2 version of its ANSI-77 FORTRAN, Lahey offered a PC ANSI-77 FORTRAN for $95, and Prospero had an ANSI-77 FORTRAN for GEM. A number of new Modula-2 implementations or versions were announced, including a new version of the well-regarded Logitech compiler, new FTL versions for several machines, and OXXI's Benchmark Modula-2 for the Commodore Amiga. California Software Products has ported RPG II to PCs. And Borland unloaded a number of language announcements, including Turbo Pascal 4.0.

Meanwhile amid the flotsam and jetsam of the show, I kept sighting ex-editors of ex-programmers' magazines who had jumped ship as their magazines sailed into strange waters. Finally, heading for the pressroom one last time, I ran into Scott Mace, who, having been with *InfoWorld* since before I moved to the Coast, must be the longest-tenured scribe for any personal computer industry publication. A real journalist and a survivor.

And Ben was in the pressroom, the stitches now out, working on his notes.

Michael Swaine

Michael Swaine
editor-in-chief

How A C Programmer Became A Screen Star

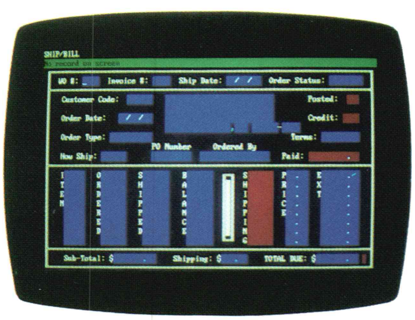
Screens, the Visible Part of Your Program.

A program is often judged by how well the screens are executed. However, the real creativity lies in what goes on behind the screens.

ScreenStar is a product that allows your real creativity to light up the screen. It reduces costly screen, window, and data validation development time.

You Take the Bows, We Write the Code.

Our natural drawing commands allow you to paint any screen imaginable. Press one key when you are satisfied and ScreenStar produces concise, commented, ready-to-compile code. This allows immediate testing of the I/O screens, including smooth, even scrolling between multiple screens.



Create or capture complex screens with data-entry filters built in.

If all ScreenStar did was turn screens into code it would be a useful tool. Yet ScreenStar also permits a wide range of field types. Some of the choices include date, alphanumeric, telephone, yes/no, dollar, time and user-definable fields.

Other valuable data-entry filters are built in, such as required field, display only, and many others. All screen fields are generated with error-checking routines.

ScreenStar Not Only Captures Your Imagination, It Captures Screens.

The memory-resident capture program converts any screen into a ScreenStar file in seconds, including those generated by programs like Dan Bricklin's Demo Program.

ScreenStar Sets the Stage for Windows.

ScreenStar comes with a complete window generating library. You design the help screens and pop-up windows. Essential ScreenStar windowing functions tie them together in one smooth package.

Curtain Call.

They may not ask for your autograph, but they will want to know how you did those screens. Screenstar is more than a screen-painting program. It is a screen processor. No professional programming environment will be complete without this product.

We know you will enjoy using ScreenStar. However, should you give it less than rave reviews, return it within 30 days for a full refund.

★ Interactive screen painting and subsequent code generation.

★ Multiple screen design and scrolling.

★ TSR screen capture program, works with any program including Dan Bricklin's Demo Program.

★ Complete window design including overlapping window functions.

★ Screens are compressed into data structures, and remain a permanent part of the program. No messy data files to look for.

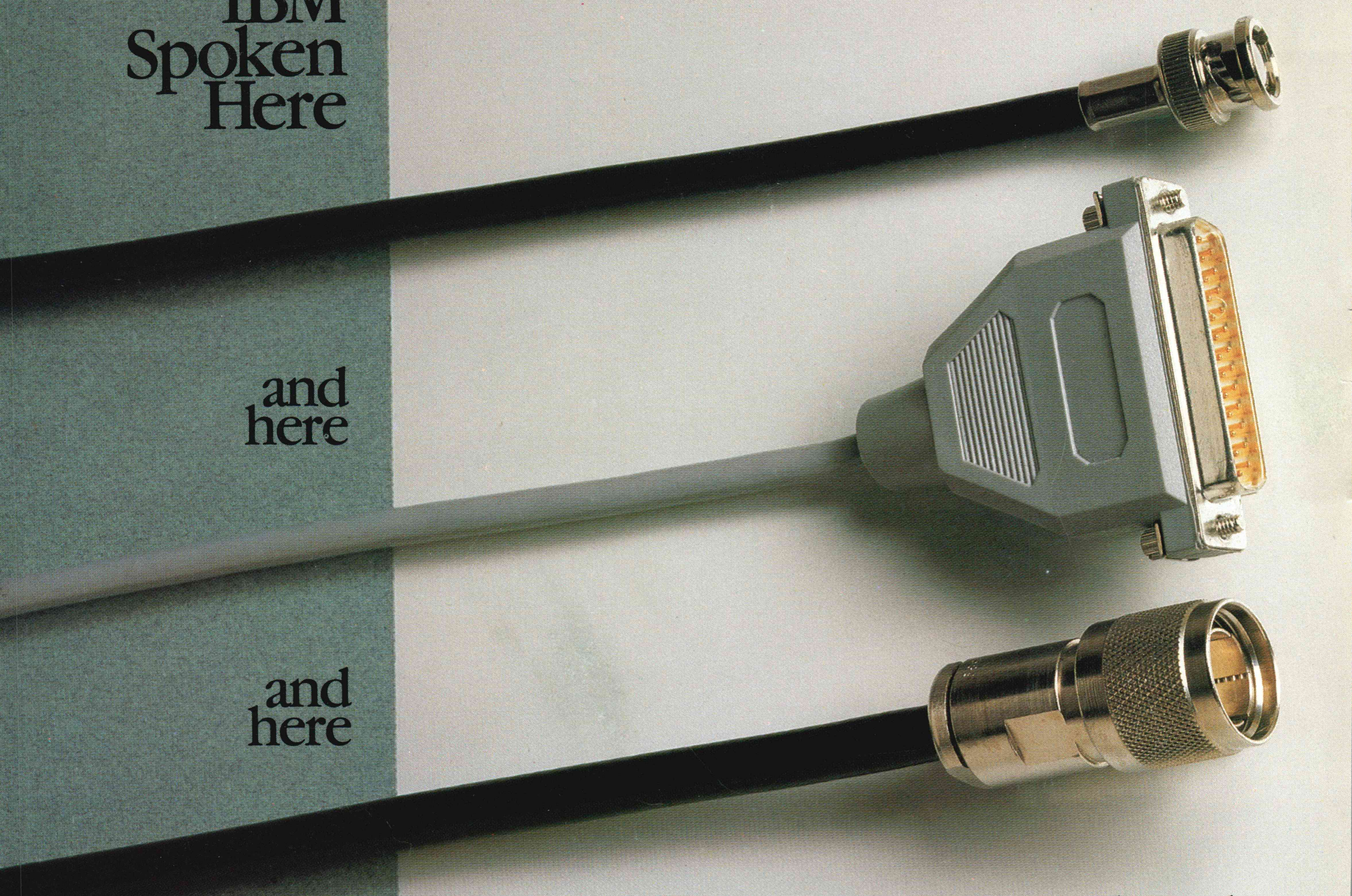
Price - $99

W/Source add $99

Audition Our Product Today. Call:

CIRCLE NO. 199 ON READER SERVICE CARD

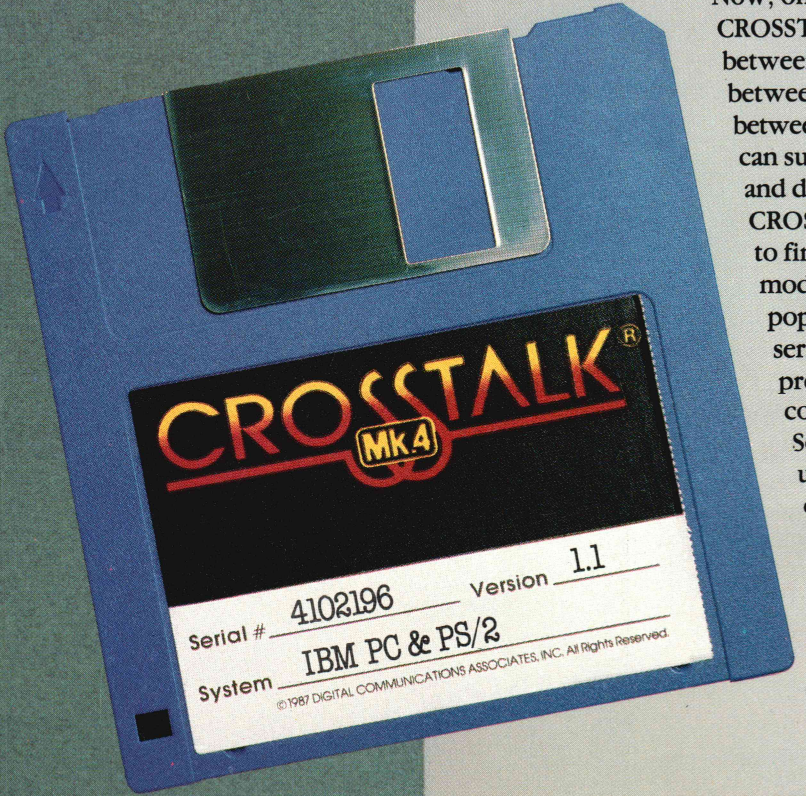
IBM Spoken Here



and here

and here

and here



Whatever dialect of IBM you need to speak, CROSSTALK® Mk. 4 makes the connection.

Now, one program does the job that used to require several. CROSSTALK® Mk. 4 allows high-speed direct communications between PCs and minicomputers, or (with an IRMA™ board) between your PC and an IBM Mainframe, or (with Smart Alec™) between your PC and IBM System 3x's. If you like, CROSSTALK can support all of these sessions (and others) simultaneously, and display each session in its own window.

CROSSTALK Mk. 4 emulates all the terminals you're likely to find useful. That includes IBM 3101 (page and character modes), IBM 525x, IBM 529x, IBM 327x, as well as many popular async terminals like the DEC VT100 and VT220 series. CROSSTALK Mk. 4 includes the powerful CASL™ programming language, which allows you to automate communications applications quickly and easily.

So if you're used to thinking of CROSSTALK just to use with a modem, you're missing some important connections. Ask your dealer for details, or write:

dca® Digital Communications Associates, Inc.
1000 Holcomb Woods Parkway / Roswell, Georgia 30076
1-800-241-6393

CROSSTALK® COMMUNICATIONS

CIRCLE NO. 200 ON READER SERVICE CARD